The Thesis Committee for Alexander Richard Tomkovich
Certifies that this is the approved version of the following Thesis:

Learning Coherent Narratives from Text-Based Knowledge Graphs

APPROVED BY
SUPERVISING
COMMITTEE:

Katrin Erk, Supervisor

Greg Durrett

Learning Coherent Narratives from Text-Based Knowledge Graphs

by

Alexander Richard Tomkovich

Thesis

Presented to the Faculty of the Graduate School
of The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

August 2020

Acknowledgments

We acknowledge the contributions of our fellow UTexas AIDA team members. Most critically to our work, we thank Su Wang for passing on his initial graph salad data generation and modeling scripts and for building the fundamental concepts behind our task and model. We thank Pengxiang Cheng and Eric Holgate for their work in preprocessing for the AIDA task. We also thank Pengxiang Cheng for his invaluable advice on data generation, modeling, and experimental setup. We thank Clara Cannon for her help as a project partner during our implementation of the reinforcement learning portion of this work for our reinforcement learning class. We similarly thank Scott Niekum and Peter Stone for their advice during our exploration of an RL framework. Lastly, we thank Katrin Erk for her invaluable guidance and insightful feedback throughout the entirety of this project.

This research was supported by the DARPA AIDA program under AFRL grant FA8750-18-2-0017. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA, DoD, or the US government. All experiments were conducted on the Chameleon testbed supported by the National Science Foundation on compute resources provided by the Texas Advanced Computing Center and the University of Chicago.

Abstract

Learning Coherent Narratives from Text-Based Knowledge Graphs

Alexander Richard Tomkovich, M.S.Comp.Sci.

The University of Texas at Austin, 2020

Supervisor: Katrin Erk

Although efforts to reason over large-scale knowledge graphs have continued to gain traction over the last decade, document-level knowledge graphs still remain underexplored. In this work, we build on previous efforts in narrative cloze and one-class clustering to develop a system which takes in as input mixtures of text-based knowledge graphs that meet at shared entity/event nodes and aims to expand subgraphs which correspond to coherent narratives within those mixtures.

We develop two main types of synthetic data with two associated tasks: "graph salads" are mixtures of text-based KGs generated from distinct documents which require our inference models to iteratively add a sequence of graph pieces to a starting narrative, while "cloze-style" instances are mixtures of text-based KGs from distinct documents which require the admission/rejection of a single graph piece. We develop corresponding neural models which leverage graph convolutional networks and an attention mechanism to perform inference over these graphs. We show that our data is learnable, and offer suggestions for future improvement of both our data generation procedures and model architectures.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Stories generally have some level of continuity to them; narratives tend to flesh out the same set of entities, events, relationships, and associations in a consistent and meaningful way. Walter Fisher first formulated this idea of "narrative coherence" in his "narrative paradigm" (Fisher, 1984). In his paradigm, Fisher proposes the metaphor *homo narrans*, or "storytelling human." In essence, Fisher argues that we give structure and meaning to the human experience by telling stories.

Fisher's narrative paradigm posits (among other points) that rationality is a product of the idea of *homo narrans*; our rules for logical judgment are informed by "narrative probability" and "narrative fidelity," which suggest that humans have an inherent notion of what makes a story coherent, and that that notion is informed by history, culture, and our individual experiences. The idea of narrative coherence, then, suggests that stories can be judged as sensical/nonsensical to varying degrees because we are uniquely suited as humans to access world knowledge, reasoning capabilities, and personal stories which inform our narrative expectations.

Schank and Abelson employed hand-written *scripts* (or textual sequences of events which occur in a specific context) in their investigation into how humans store, access, and leverage knowledge; in exploring why coherent scripts are coherent, they identify a set of qualities (causuality, plans, themes, etc.) which reinforce the idea that humans, when writing, translate their capacity to reason into a set of recognizable patterns which make stories coherent (Schank and Abelson, 1977).

Because we (therefore) assume that humans write texts which tell coherent stories, we hypothesize that it is possible to machine-learn the idea of narrative coherence from texts. A number of works support this theory; Chambers and Jurafsky, for instance, successfully order and cluster chains of events by leveraging both syntax- and semantics-level features of text (Chambers and Jurafsky, 2008).

In this work, we explore how this idea of narrative coherence can be leveraged to make sense of complicated real-world stories via trained neural networks. We describe our work on the Active Interpretation of Disparate Alternatives (AIDA) project, a DARPA-funded research effort which aims to stimulate the development of automated systems which can help reporters and decision-makers more quickly understand the

multitude of potential explanations as to how a particular geopolitical event played out.

In our work, we represent narratives as *knowledge graphs* which describe, ideally in mutually conflicting ways, how a particular story's key actors and events relate to each other. Our task is the following: given a knowledge graph which is a mixture of distinct stories that share common entities/events, and given a subgraph that represents an initial piece of one of those stories, we seek to expand that subgraph by adding other pieces of the graph which are coherent with that seed narrative. Our system is a neural model that develops contextualized representations of these so-called "graph salads" via a graph convolutional network and works to expand a small starting "seed" subgraph in each salad which represents the initial kernel of a distinct story.

Because there are no existing gold-label training sets, we develop accompanying data which simulate a multi-narrative environment by artificially merging distinct Wikipedia documents. We present two main flavors of data: a set of graph salads for use with non-contextualized word embeddings, and a set of so-called "cloze-style" instances for use with contextualized word embeddings produced by a pretrained ELMo model; we motivate the need for two separate data sets after theorizing that simply injecting contextualized embeddings into our "graph salads" task may lead to trivial inference. We show that our synthetic data is learnable and go on to examine opportunities for future improvement in both our data generation and modeling efforts.

1.1 Related Works

1.1.1 Narrative Cloze and Coherence

Drawing on the work of Schank and Abelson, Chambers and Jurafsky later show that scripts can be learned from data. Specifically, they introduce the idea of *narrative event chains*, which are used in a *narrative cloze* task that requires inference models to "fill in" missing details from sequences of events (Chambers and Jurafsky, 2008), and they leverage coreference chains to learn these scripts. Chambers and Jurafsky later generalize their narrative event chains to "narrative schemas," modeling both arguments and events to aid inference in cloze tasks (Chambers and Jurafsky, 2009).

Our work most directly continues the efforts in (Wang et al., 2018) and (Wang et al., 2019). In (Wang et al., 2018), the authors cite inspiration from works exploring conversation disentanglement (Elsner and Charniak, 2008, Pennington et al., 2014, Jiang et al., 2018) and document distinction (Bekkerman and Crammer, 2008) tasks and introduce a "story salad" data set which combines sentences from different narratives and asks inference models to cluster (Bekkerman and Crammer, 2008) sentences from the same narrative given an initial "seed."

Wang et al. continue to develop these ideas in (Wang et al., 2019), in which they create synthetic data composed of sentences which feature events that may contradict each other; they task a neural model with clustering "compatible sets of events" together. This inspires our "iterative statement admission" approach (described in **Section 3.2**).

1.1.2 Knowledge Graphs

The application of machine learning principles to inference over knowledge graphs (KGs) has seen considerable interest from the computational linguistics community in the last decade (Lao et al., 2010; 2011, Bordes et al., 2013, Yao et al., 2013, Dettmers et al., 2017, Das et al., 2017). Knowledge graphs are settings which are uniquely suited to the flexible representation of information; the simple yet powerful compositionality of graphs means that long-range relationships like the semantic role labels of arguments with respect to events can be easily distilled and represented in a graph setting (Marcheggiani and Titov, 2017), even if the complexity and number of those relationships is high.

Efforts in the task of link prediction (a setup similar in nature to cloze-based inference in which a model infers new edges between nodes in an incomplete knowledge base), in particular, have shown that a graph-based paradigm is a particularly effective setting for inferring relationships between entities in KGs (Lao et al., 2010; 2011, Das et al., 2017, Xiong et al., 2017).

1.1.3 Graph Convolutional Networks

Similarly, the spatial nature of knowledge graphs allows for an accessible way to propagate information; graph convolutional networks (Kipf and Welling, 2017),

or GCNs, serve as intuitive first-order ways of contextualizing nodes by virtue of their connections to other nodes in a KG. First introduced by Kipf and Welling as a means of approximating spectral graph convolutions (Hammond et al., 2009), graph convolutional networks propagate information throughout a graph structure by performing localized, filter-like convolution operations on neighborhoods of graph nodes. It is through these convolution operations in our model's GCN that we are able to contextualize events and entities by their surrounding associations.

Investigations into leveraging pathing information between nodes and traversing graph structures have similarly enjoyed the benefits that a structured spatial setting provides (Lao et al., 2010; 2011, Das et al., 2017, Chen et al., 2019); a well-known issue with GCNs is their inability to capture longer-range dependencies in graphs (Marcheggiani and Titov, 2017, Chen et al., 2019).

While Marcheggiani and Titov made early efforts to remedy this issue in a semantic role labeling setting by supplementing GCNs with an LSTM network (Marcheggiani and Titov, 2017), more recent efforts have taken inspiration from the "transformer" paradigm (Vaswani et al., 2017). Veličković et al. move toward a purely self-attentive network which is able to implicitly assign weights to different neighbor nodes (Veličković et al., 2017); similarly, Chen et al. extend this transformer-based approach by allowing nodes to attend globally to all nodes in a graph structure via attentive operations informed by shortest-path features between nodes (Chen et al., 2019).

While we currently employ a fairly simple GCN in our models which limits our ability to observe and leverage longer-term dependencies between nodes, the recent bevy of literature in attentive and path-based methods promises to be a compelling avenue for future models for our task.

1.1.4 Reasoning Over Text-Based Knowledge Graphs

While reasoning over KGs more generally has enjoyed substantial exploration, reasoning over KGs *extracted from text* have seen smaller efforts (Zelinka et al., 2019); the amount of processing that raw texts go through during their translation to knowledge graphs often results in a substantial loss of information, thus making KGs non-ideal for inference over text-based sources.

Because the AIDA project emphasizes the use of KGs where are drawn from a

variety of multilingual and multimodal sources, however, knowledge graphs represent a unifying form of representation which is appropriate for inference over even text-based sources in this case. In this work, we elect to explore whether we can leverage the unique properties of graphs to perform inference over knowledge graphs extracted from text.

1.1.5 Information Extraction

Unfortunately, the relative simplicity of graph structures means that raw texts must undergo a fair amount of preprocessing before they can be represented via KGs. The task of *information extraction* fills this need by analyzing text and formulating key elements into structured representations (Singh, 2018). Of particular relevance to our task are entity and event extractors (Li et al., 2013, Judea and Strube, 2016, Li et al., 2019, Lin et al., 2020), which (as one might expect) analyze text with the goal of identifying entities and events and defining their relationships (e.g., that a particular person is the victim of a killing).

1.1.6 Reinforcement Learning

Lastly, RL has proven to be a prolific setting for the exploration of graph-based inference. RL-based methods have made significant strides in (as previously referenced) the tasks of link prediction (Das et al., 2017) and multi-hop reasoning (Xiong et al., 2017, Lin et al., 2018) in knowledge graphs, both of which require the kind of sophisticated entity-based reasoning our task demands.

In You et al., 2018, the authors construct synthetic molecular graphs by adding structured graph pieces one-by-one to a growing subgraph (You et al., 2018). They represent the current state of the environment as the intermediate graph at a given time step, and the action space as the set of possible edges to new subgraph pieces.

Given the applicability of RL-based methods to graph environments, we try framing our task in an RL setting which is most similar to that in (You et al., 2018); we iteratively expand an initial subgraph narrative by adding other pieces of a graph mixture one-at-a-time.

Chapter 2

Task Preliminaries

In this chapter, we review the fundamentals of our task, including our role in the AIDA project and the structure of the knowledge graphs we operate on.

2.1 The AIDA Project: Graph-Based Reasoning over Competing Narratives

In this section, we briefly shift our focus in order to introduce the high-level details of the AIDA program as they relate to efforts to advance reasoning over knowledge graphs.

2.1.1 Task Motivation

Our task motivation is a continuation of that in (Wang et al., 2018) and (Wang et al., 2019); events of geopolitical interest often unfold in rapid and complicated fashions, and can lead to the creation of diverse pieces of media which offer differing perspectives on how things played out. If a plane is shot down, for instance, the following day might bring reports on the attack from 15 distinct news outlets, a number of eyewitness videos of the event, a collection of pictures of the downed aircraft, etc. Given the short time in which they are often generated (and given the sheer number of sources which offer perspectives), these pieces of media can be difficult to disentangle when it comes to identifying and understanding key details of what happened during an event. Reports covering the aforementioned attack on a plane, for instance, might differ in their characterizations of what weapon was used to carry out the attack, who perpetrated the attack, how many individuals were on board, etc. to varying degrees of contradiction.

The Active Interpretation of Disparate Alternatives (AIDA) project aims to generate systems which are capable of piecing together coherent narratives from a collection of diverse media covering the details of a geopolitical event. The AIDA workflow pipeline is separated into three stages: TA1 groups perform information extraction on raw media (including relevant texts, images, and videos) and produce single-source

knowledge graphs, TA2 groups perform cross-document coreference resolution and merge these single-source graphs into multi-narrative graphs, and TA3 groups traverse those graphs and identify coherent narratives. During a yearly evaluation period, involved teams are asked to run their pipelines on a selection of media pieces which represent narrative takes on elements of a particular geopolitical issue (e.g., the Russia-Ukraine conflict).

2.1.2 Our Team's Focus

As a TA3 group, the UTexas team is provided with knowledge graphs produced by TA2 teams and asked to identify coherent narratives within those graphs, given a focused subset of starting information. Given a knowledge graph covering a set of related attacks during the Russia-Ukraine conflict, for example, we might be told that a building in a particular city was involved in some kind of fire. After locating a building entity and city which we believe are the ones hinted at (and perhaps also locating a suitable node that might correspond to the associated conflict event), our task is to iteratively identify and add to the story other pieces of the graph which further explain details regarding what happened. We refer to this process of fleshing out a fledgling story one detail at a time as "iterative narrative expansion."

Because we do not have access to any gold-label data for the AIDA task, we develop our own sets of synthetic data which we present in **Chapters 3** through **5**.

2.1.3 AIDA Wrap-Up

Now that we have covered the high-level particulars of the AIDA task, we avoid further details of the AIDA paradigm whenever possible in the rest of this work, instead choosing to frame our task more generally as inference over text-based knowledge graphs.

We make one final preliminary note that the example graphs shown in later figures are illustrated through the use of a tool we developed using the `dash-cytoscape` graph visualization environment, accessible at `https://pypi.org/project/dash-cytoscape/`.

2.2 Knowledge Graph Structure

In this section, we review the structure of the knowledge graphs on which we operate.

2.2.1 Events, Relations and Entities (EREs)

In our task, narratives are broken down into compositions of three key pieces of information: "events," "relations," and "entities" (hereafter collectively referred to as "EREs").

Events are goings-on contained within a story which are supplemented by one or more "arguments" which play a role in those goings-on. Some example events are as follows, along with some of their possible associated arguments: an ambush during a war (*args*: the attacking group, the group being targeted, the instrument used to carry out the ambush); the transportation of an official to a military base (*args*: the person being transferred, the destination, the vehicle involved); a bank transaction (*args*: the giver of the funds, the recipient).

Entities are people, places, or things which assume roles with regard to one or more events or relations (described shortly). The role of the "attacking group" entity in the "ambush" event, for instance, would be that of "attacker." The arguments given along with the examples above are all designated as entities. More generally, all arguments to events or relations are entities, and all entities are arguments to events or relations.

Lastly, **relations** are associations between entities; a relation may specify, for instance, that a military base is located near a particular city, or that a son and mother are family members. Relations, like events, take entities (and only entities) as arguments. As an additional constraint, relations (unlike events) must have exactly two arguments.

Examples of an event and a relation are presented in detail in **Sections 2.2.4** and **2.2.5**, respectively; we first introduce statements and ontology labels, however.

2.2.2 Statements and Ontology Labels

In order to succinctly compose these pieces of information in a structured setting, we operate on knowledge graphs (hereafter "KGs"). A single knowledge graph offers

information on a collection of related EREs.

EREs involved in a narrative are represented by nodes in a knowledge graph. These nodes are connected by directed edges which represent "statements" specified under the narrative. *Statements* describe the roles which entities take on in specific events or relations. In the "ambush" scenario mentioned in **Section 2.2.1**, for instance, a statement would link the ambush event node and the entity node that represents the attacking group and indicate that that entity took on the role of "attacker" in that ambush.

Lastly, each ERE and statement is assigned an *ontology label* which designates its classification under the AIDA ontology. In the "ambush" scenario above, the attacking and targeted groups might be classified as "GeopoliticalEntities" (if the corresponding referents in the text are countries) or "Organizations" (if specific infantry squadrons are specified), and the entity node which was used to carry out the attack might be classified as a "Weapon."

Under this ontology, periods indicate a narrowing of specificity (e.g., for the label "Conflict.Attack," the word "Attack" is in a more specific class than the word "Conflict"), and underscores precede the roles that entities play in events or relations (e.g., in the upcoming **Figure 2.1**, "the Yokosuka Naval Arsenal" takes on the role of "Destination" in the shipping event). For the sake of discussion, an ontology label can be conceptualized as an additional descriptor or field which accompanies each ERE node in a KG. When referring to ontology designations, we use the terms "label" and "type" interchangeably.

2.2.3 ERE Representation

It is important to note that an event or entity is represented using two fields: a set of *names* (or *mentions*) and an ontology label. *Names* are the textual spans attributed to referents in a source text; in **Figure 2.1**, for instance, the entity to the left of the shipping event was referred to using the word "submarines" at some point in the text. Independent of this name is the entity's ontology label ("vehicle"), which offers a general suggestion of its qualities as a narrative participant. Note that entities and events can have more than one name, as they may have more than one coreferent in the source text.

Although relation nodes also have both a name and ontology label, the textual

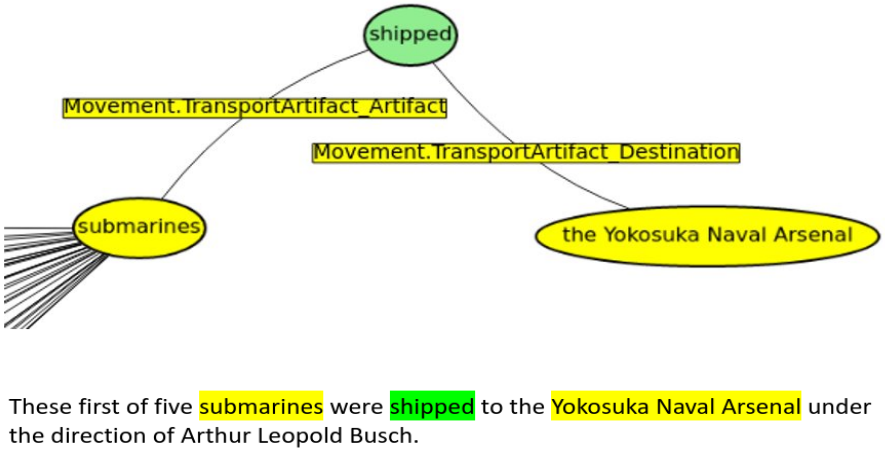

Figure 2.1: Example event

span which constitutes a relation's "name" (in practice) tends to consist of a full sentence from text (e.g., "She was the daughter of the deceased monarch.")

2.2.4 Example Event

An example of an event node (along with its corresponding arguments) can be seen in **Figure 2.1**; here, we present an event classified as a "Movement.TransportArtifact" event under the ontology. The word "shipped" represents the name of the event node. The entity being moved ("submarines") and the entity for which the transport is destined ("the Yokosuka Naval Arsenal") are similarly identified, with their respective roles ("Artifact" and "Destination") in the TransportArtifact event represented in the labels following the underscores in the attached statements. The source text from which the event and its two arguments were pulled is also pictured. Ontology labels are not shown for the two entities pictured.

2.2.5 Example Relation

Similarly, an example of a relation node and its arguments is pictured in **Figure 2.2**. Here, a "Physical.LocatedNear" relation is shown with arguments "airport" and "Bridgeport," capturing the associated relationship embedded within the text. The relation node's ontological type ("Physical.LocatedNear") is also pictured. Although not shown, "airport" is typed as a "Facility," and "Bridgeport" as a "GeopoliticalEn-

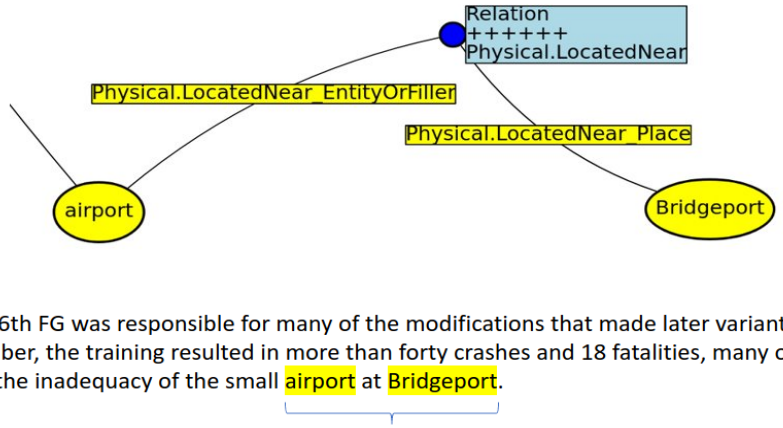

Figure 2.2: Example relation

tity."

Chapter 3

Data Set 1: Graph Salads

In the remainder of this work, we overview our three "flavors" of synthetic data and our efforts to reason over them. Each "flavor" of data is motivated, described, and employed in an inference task separately in each of **Chapters 3** through **Chapter 5**; we follow this format for the sake of continuity and clarity.

3.1 Task Setup

Before launching into our first type of synthetic data (and associated model architectures), we take a moment to discuss how we translate our team's interests in exploring graph-based reasoning into a practical, machine-learnable task. Recall (from **Chapter 2**) that our principal goal is to investigate graph-based reasoning over knowledge graphs drawn from text-based sources, and that geopolitical happenings often yield confusing and competing stories. Given these two points, we amend our earlier investigative goal from **Section 1.1.4** to be, "Can we discern via a trained neural model between stories which are drawn from singular textual sources and represented as knowledge graphs?" In order to begin addressing this question, we need (i) a setting in which stories from multiple documents populate a single combined knowledge graph and (ii) a way of systematically traversing a narrative in graph form.

Our first paradigm addresses these needs by formulating inference as a process of "iterative narrative expansion" within the context of what we refer to as "graph salads," described in detail in the sections below.

3.2 Iterative Statement Admission

Before beginning a review of our data generation process, we first give a quick summary of the inference procedure we ask our models to carry out. Assume that we are given a large KG which is constructed by merging smaller KGs representing different stories at a subset of EREs ("merge points"). Assume we are also given a couple of statements which we are told represent the initial kernel of one of these

distinct stories. Our task is to add other statements in the knowledge graph which cohere with the seed narrative one-by-one. This process takes inspiration from the work in (Wang et al., 2019), in which the authors use an initial "query" set of events to inform the creation of a growing "scenario-in-construction."

We refer to the initial kernel of a story as our *query set*. The set of statements which were either (i) part of the initial query set or (ii) added to the expanding subgraph during inference comprise the *narrative-so-far* subgraph. At a given step in a sequence of admissions, the set of all statements which (i) share an ERE with any statement in the narrative-so-far subgraph and (ii) have not yet been added to the growing narrative constitute our set of *candidate statements*. For each admission step, our goal is to use the narrative-so-far subgraph to inform which candidate statements we add to the narrative.

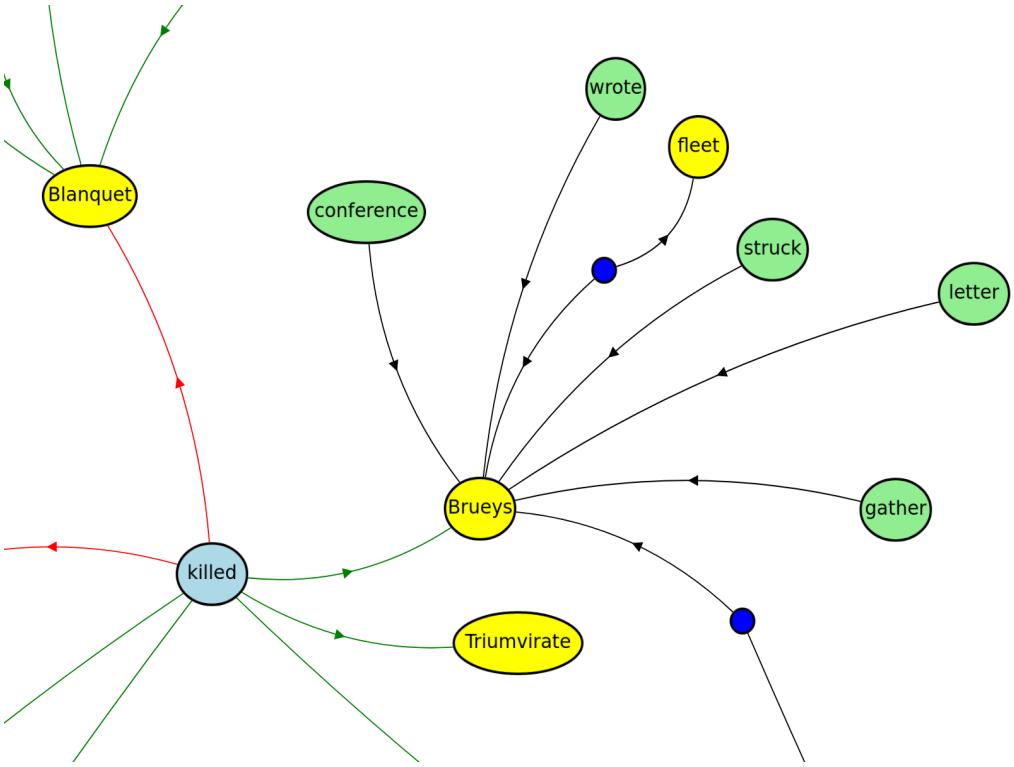

Figure 3.1: Before statement admission

Examine **Figure 3.1**. In this case, the two red statements surrounding the "killed" event comprise our query set; in particular, for example, we know at the start of inference that the entity "Blanquet" was a victim of a killing. All green statements

represent our candidate statements (the set of possible statements we can admit at this particular time step). Assume we decide to admit the statement identifying the entity "Brueys" as another victim of the killing. After adding this statement to our narrative-so-far, our inference state updates to that pictured in **Figure 3.2**; i.e., we are now allowed to admit the other statements immediately surrounding the "Brueys" entity.

This strategy of admitting statements from only the immediate "fringe" around our narrative-so-far allows us to limit our search space at a given time step; we need only consider the statements which are immediately next to our narrative-so-far subgraph, since our goal at the end of expansion is a contiguous subgraph.

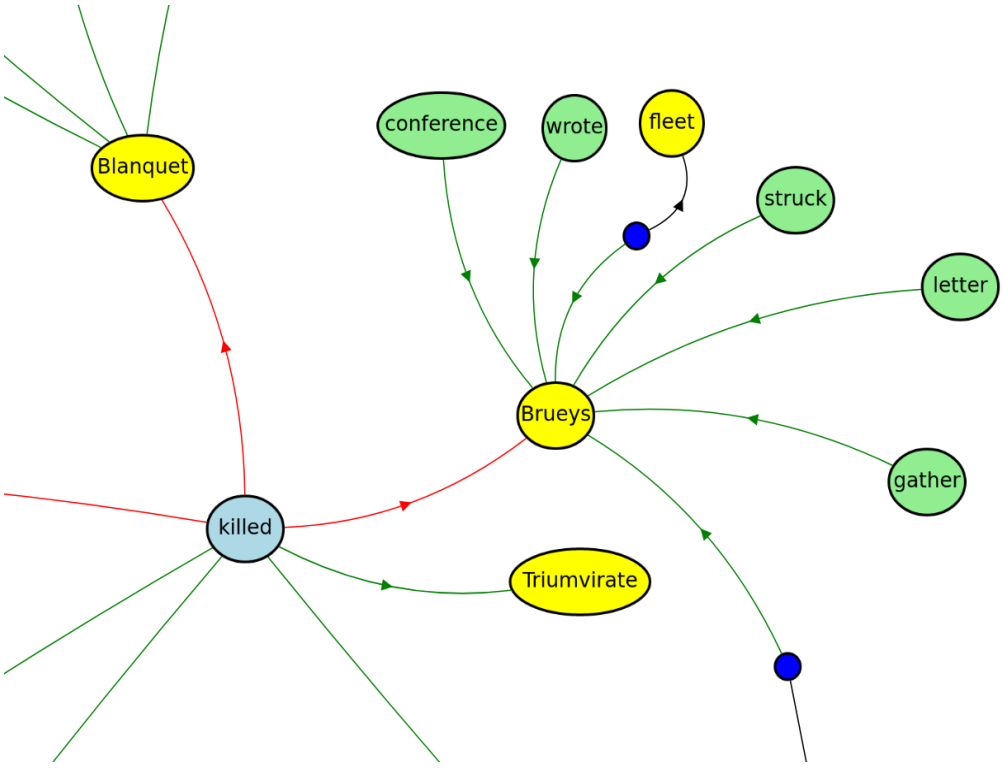

Figure 3.2: After statement admission

3.3 Graph Salad Generation

As suggested in (Wang et al., 2019), a key characteristic of a challenging multi-narrative data set is the presence of often contradictory or mutually incompatible narrative pieces in a single knowledge graph. Assume, for instance, that we are

examining a graph which outlines the details surrounding an event in which a plane is shot down (a "Conflict.Attack" event). This knowledge graph will likely be composed of pieces derived from source materials which offer competing characterizations of how the event unfolded; one story might suggest that insurgent forces attacked the plane with surface-to-air missiles, while a second one might suggest that another aircraft belonging to a third-party nation carried out the attack. In either case, key details related to the attack (namely, the weapon/instrument used and the attacker involved) must be disentangled and resolved when constructing a singular narrative to characterize the event.

3.3.1 Data Sources

Needless to say, it is difficult, if not impossible, to manually create a data set of sufficient scale that adequately addresses this need for collections of narratives that are topically similar, but factually diverse. To address this issue to the best of our ability via synthetic means, we build on the "story salads" approach introduced in (Wang et al., 2018), wherein mixtures of narratives are artificially generated by interweaving sentences from pairs of topically similar articles into a single body of text.

The idea underlying the approach taken by Wang et al. is that sentences which tell different stories should be distinguishable in a classification task. This fundamental idea similarly motivates our approach; we extend the "story salads" approach by artificially merging subgraphs from distinct source documents into larger, multi-document KGs, which we refer to as "graph salads," following (Wang et al., 2018).

Wang et al. experiment with constraints intended to induce challenge in their data, including the use of a topical similarity score between source texts. They similarly construct a dataset of 50k salads which draw from Wikipedia articles whose associated category designations contain the words *conflict* and/or *war*; they find that these categorical constraints help to produce a challenging dataset that requires nuanced inference. Given that our realm of interest is geopolitics (which often manifests itself in conflict-related events), we follow their lead and draw our source texts from this same set of 81,022 *conflict*/*war* articles.

3.3.2 Graph Salad Preliminaries

We use the information extraction tool in (Li et al., 2019) to convert 72,167 of the 81,022 Wikipedia articles referenced above into individual knowledge graphs. (Note that the parser was not successful in producing output for all the Wikipedia articles.) Of these 72,167 KGs, we find that only 68,666 KGs have at least one statement linking EREs, with the others simply consisting of sets of unconnected ERE nodes. Hence, the final size of the set of usable source Wikipedia KGs which we draw from is 68,666.

The parser also records the character offset indices corresponding to each ERE mention as it appears in the source document; we later make use of these during our setting with contextualized word embeddings. For instance, in **Figure 3.2**, we have access to information telling us that the name "Triumvirate" appears in, say, characters 811-821 in the source document.

3.3.3 Source Graph Selection

Now that the raw Wikipedia texts have been converted into KGs, we apply the "story salads" approach and combine these distinct source graphs into multi-narrative KGs. We refer to these multi-narrative KGs as "graph salads," following (Wang et al., 2018).

Selection of Source KGs

Since our ultimate goal is to distinguish between narratives when multiple competing options are presented to us, it is necessary that the KGs we operate on be composed of at least two or more component graphs. We combine KGs by artificially merging single nodes from individual source graphs into so-called "mixture points."

The decision to set the particular number of KGs mixed for each graph salad presents an important tradeoff. The density of statements (or narrative options) around mixture points, for one, increases dramatically when additional KGs are used to create a mixture. However, as the number of component graphs used to create a mixture rises, it becomes increasingly difficult to find groups of KGs which meet the criteria (discussed in **Section 3.3.4**) for mixing. As a balance between these two considerations, we settle on combining a total of three component graphs per salad.

We note that a few other key design choices made in **Sections 3.3.4** through **3.3.6**

Source Doc.	Ontology Type	Name Set	Connectedness
Event A	Conflict.Attack	{“strafing”, “firing”, ...}	13
Event B	Conflict.Attack	{“strafing”, “fire”, ...}	7
Event C	Conflict.Attack	{“strafing”, “raids”, ...}	11

Table 3.1: Merge candidate characteristics

are justified in **Section 3.3.7**; we feel that these choices are more easily understood and rationalized after we have walked through a specific example of a generated salad, and so we table discussion of these choices for a moment.

3.3.4 Graph Triplet Selection

Entity/Event Matching

To begin creating a graph salad, we first identify a set of three Wikipedia source graphs (hereafter, a “triplet” of graphs) which have in common a minimum of three events/entities which (i) share the same ontological type and (ii) have some overlap in their names, where the term “names” is shorthand for the textual spans associated with a node.

See **Table 3.1**. Assume we examine a triplet of three source graphs (A, B, and C) as potential candidates for mixing. If we identify an event node in each graph (say, Event A, B, and C) whose ontology type is “Conflict.Attack” in each case, and which contains the word “strafing” in the list of names associated with the event, then Events A, B, and C are considered mergeable.

If we similarly identify two other triplets of EREs in graphs A, B, and C (say, Entities D-F and Events G-I, with shared names “aircraft” and “attacked,” respectively) whose constituent nodes match in ontology label and at least one name phrase, then graphs A, B, and C can be merged to form a graph salad.

We remark on potential issues with this name-matching heuristic in **3.3.7**.

Priority by Connectedness

Among candidate entity node triplets determined to be viable for merging, we prioritize selection by each triplet’s total “two-step connectedness,” which we define as the total number of ERE’s that can be reached from a given node within two

statements of traversal (excluding the node itself). For the proposed "strafing" merge point detailed in **Table 3.1**, for instance, we sum the two-step connectedness scores for each constituent node to produce a total connectedness score of 31.

3.3.5 Merging Procedure

After identifying a triplet of nodes across three graphs which can be merged, we "collapse" the three nodes into a single representation and add all reachable statements and ERE's from each component graph into one graph "salad." This process is repeated for two other sets of nodes (entities D-F and G-I) identified as merge candidates among the triplet of graphs, with priority assigned (again) to highly connected candidates.

After a graph salad with three merge points has been created from three component graphs, up to three of the component graphs are selected to be the "target" narrative (or the gold-label narrative for that instance) in separate instances of that graph salad. We explain why we use the phrase "up to three" when discussing design considerations in **Section 3.3.7**

For the example graph salad which contains the "strafing" merge point, for instance, we might produce a total of three separate training instances: one in which the target narrative is graph A, one in which the target narrative is graph B, and one in which the target narrative is graph C. All three instances share the same KG, but differ in which component narrative we ask our model to expand.

We take this moment to summarize the criteria for a graph salad as follows:

- The salad must be composed of 3 component source graphs which have in common a minimum of three events/entities which share an ontology type and at least one name
- Preference in graph selection is given to those which maximize the number of statements around mixture points

3.3.6 Seed Narrative Construction

Once a target graph has been identified for a training instance, we follow the idea of "query-focused" inference in (Wang et al., 2019) and develop a "seed"/"query"

narrative for our model to expand. This seed narrative is constructed by doing the following for the most highly connected merge point in the graph salad:

- Add to the query set two random statements from the target graph which directly connect the merge point to a neighbor event or entity

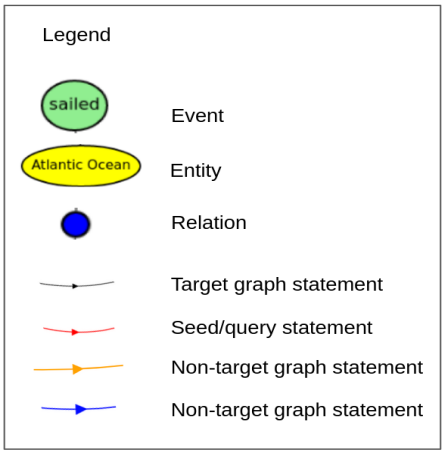

Figure 3.3: Legend for graph salad visualizations

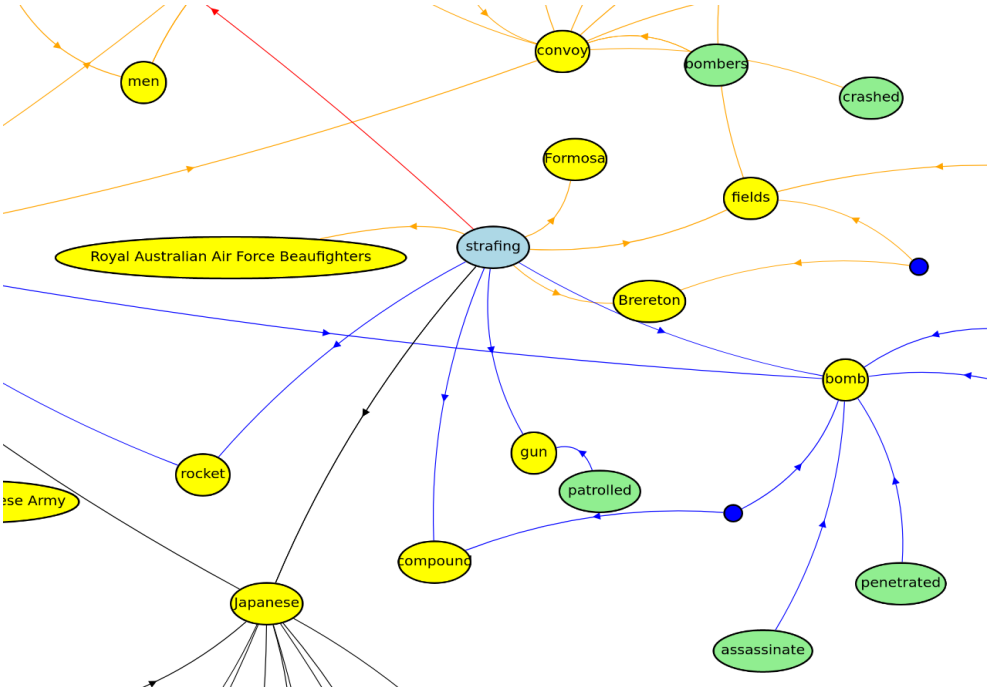

Figure 3.4: Example of merge point in graph salad

The merge point resulting from the mixture of the three constituent nodes from **Table 3.1** is pictured in **Figure 3.4**. A corresponding legend is displayed in **Figure 3.3**. Black statements are statements from the target component graph which have not yet been added to the narrative-so-far, while red statements are statements from the target graph which are provided to the model as part of an initial "seed" narrative (determined via the process outlined above). Blue and orange statements are from the two non-target graphs in the triplet.

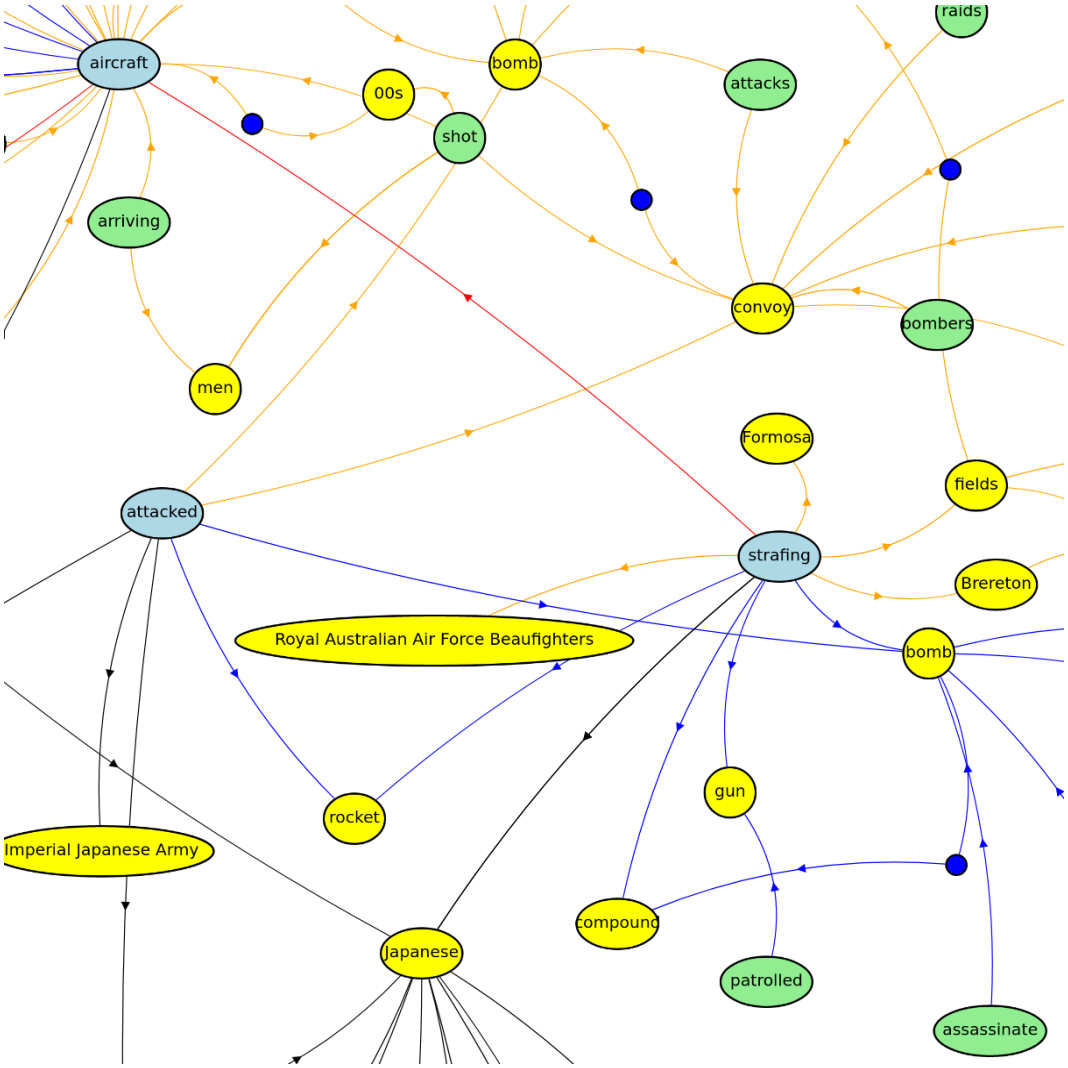

Figure 3.5: Example graph salad

A portion of the completed graph salad for the "strafing" example discussed earlier is pictured in **Figure 3.5**. The merge points/nodes are highlighted in light blue.

3.3.7 Design Considerations and Discussion

In this section, we motivate key design choices in our graph salad generation process and remark on existing problems.

Event/Entity Name Matching

As discussed during the presentation of **Table 3.1**, we require that event/entity nodes which are to be merged have at least one name phrase in common. This is a simple constraint we impose in order to enforce a base level of topical similarity among selected component graphs and improve the semantic validity of statements surrounding merge points. In the graph salad presented in **Figure 3.5**, all three component stories (an article on the Boeing B17 Flying Fortress; an article on the bombing of Saigon in 1962; and an article on the Battle of Milne Bay) involve aircraft and bombing runs.

Although this constraint leads to generally increased topical similarity among documents we select for mixing, it can produce some problems. First, entity/event name matching on unspecific nodes can lead to the mixing of component graphs of very little topical similarity; identifying a set of three graphs which have nodes with the names "building," "road," and "official," for instance, might lead to the mixing of (i) a document in which a high-ranking military official sends a tank to destroy a makeshift insurgent stronghold located near a popular road and (ii) a report on domestic wartime civil construction projects (as released by an official in urban development).

In this case, the resulting mixture of narratives would likely be easy to disentangle; the names of the entities chosen as merge points lack detail and are applicable to a wide range of scenarios, thereby decreasing the likelihood that selected component sources overlap in their most salient narrative elements. While the narratives being mixed are likely to be distinguishable (and, therefore, amenable to a classification task), matching on unspecific names might yield fairly trivial examples.

The graph salad presented in **Figure 3.5** is somewhat indicative of this issue; the three narratives being mixed are, indeed, distinct in their topical focus (and, therefore, likely able to be discerned), but the coalescence of otherwise historically unrelated events (a battle during World War II and a 1962 bombing in Saigon) at generic common events and entities fails to result in a mixture that satisfies our task's ideals

(i.e., to present a variety of narratives which concern the same geopolitical content, but which differ in the perspectives they offer on that content).

On the other end of the spectrum, matching on more specific event/entity names has the potential to produce instances with component subgraphs that are redundant in their content. Specific entities tend to be tied to more specific contexts; for example, a mention of General Pickett generally accompanies texts describing the Battle of Gettysburg. One might imagine a scenario in which two documents being mixed make reference to General Pickett and his role at the Battle of Gettysburg; in such a scenario, it is possible that the region immediately surrounding a "General Pickett" entity node in the resulting graph salad may consist of fairly redundant subgraphs from each component source summarizing information about his role at Gettysburg. The resulting salad subgraph immediately surrounding the "Pickett" merge point would likely be difficult for even a human evaluator to disentangle.

Upon review, our requirement that events and entities overlap in their name set is perhaps an unnecessarily stringent constraint; on the one hand, as long as two events/entities which are to be merged share the same ontology label, our name-matching constraint might rule out otherwise acceptable matches (e.g., an entity with the name "car" and an entity with the name "automobile;" an event with the name "battle" and event with the name "skirmish").

On the other hand, dropping the name-matching requirement might lead to some loss in semantic fidelity; should we match the entities "Hawaii" and "France" because they are both classified as geopolitical entities, the resulting KG (which will only represent the merged entity with the target ERE's name [assume it is "Hawaii" in this case]) will misrepresent Hawaii as being near the Atlantic Ocean. Whether such semantic inconsistencies would actually change the dynamic of the task at hand, however, is unclear; in particular, for events, ontology labels alone appear to seem generally sufficient to warrant matching, as most "Conflict.Attack" events (for example) tend to involve the same general themes (e.g., weapons, harm, death, etc.).

Overall, the question of how to select EREs for merging remains a difficult consideration. Given the level of nuanced understanding required to determine that two KGs cover the same geopolitical event in factually different ways, our hope of robustly simulating this particular setting via synthetic means seems far off, pending a move toward (for example) leveraging fact-checking in the future. For now, our focus

should remain on improving our ability to produce salads with reliably discernible but topically similar narratives, even if those narratives fail to present competing facts about a singular geopolitical situation.

Prioritization of Highly Connected Merge Points

As mentioned in **Section 3.3.4**, we prioritize the selection of nodes for mixing based on their two-step connectedness scores (or the total number of nodes reachable within two statement traversals). This is intended to induce regions of high density around merge points. Since the seed/query narrative is initialized using statements at merge points, we hope that this preference for density around merge points leads to the immediate visibility of diverse narrative choices to a model starting the extraction process, thereby introducing more uncertainty during inference and increasing the difficulty of the task.

While this mechanism increases the chance of achieving a large number of diverse statements around merge points, it fails to account for statement diversity in more distant regions of the graph. Observe **Figure 3.6**; the neighborhood of the entity node "Japanese" is composed exclusively of a high number of statements from the target narrative. We discuss a potential solution for this in **Section 3.3.8**.

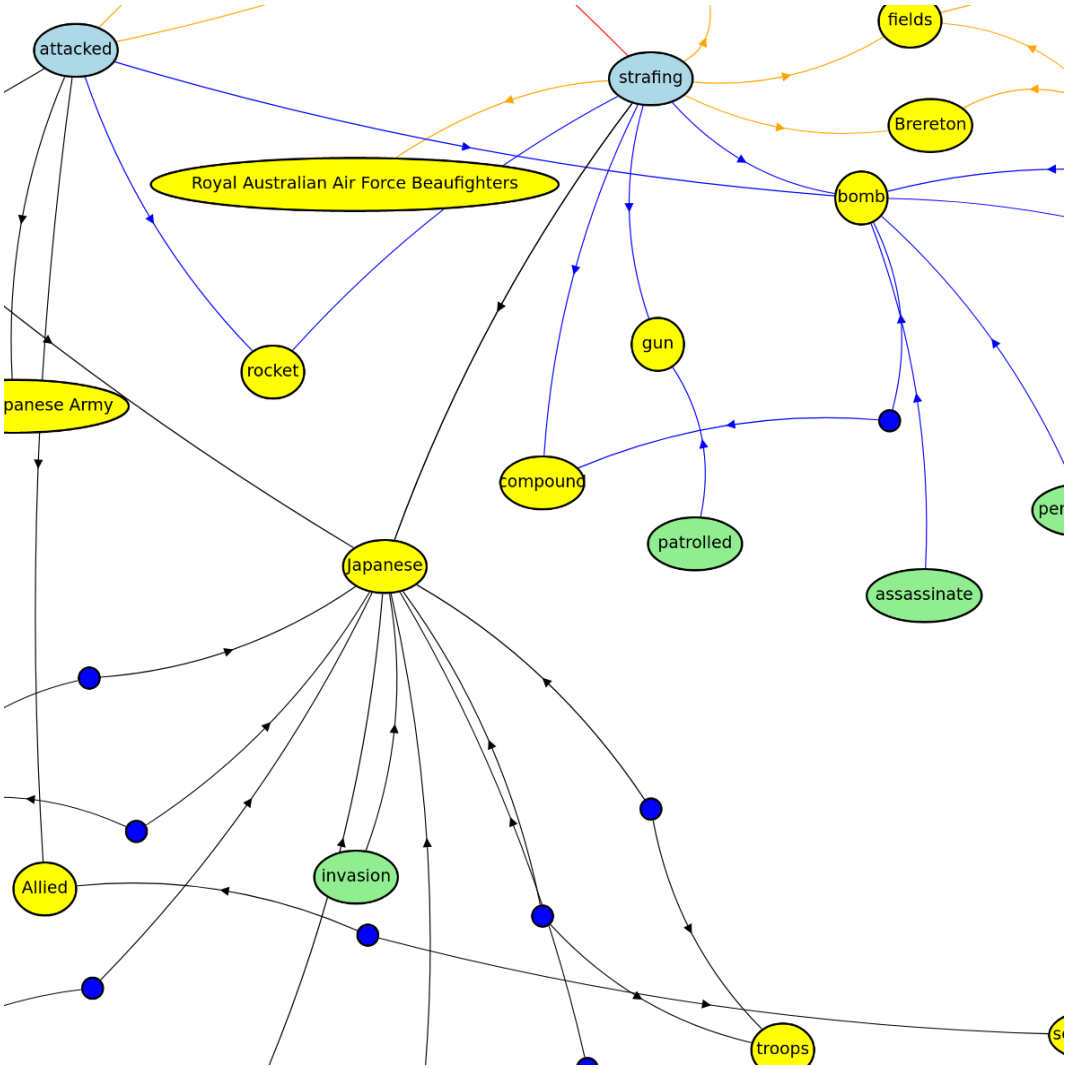

Figure 3.6: Example of less varied graph region

Selection of Target Graphs and the Reachability Constraint

Recall our remark that, for a given graph salad, "up to three" instances of that salad might be instantiated, with different component graphs identified as the target narrative in each instance.

This arose from an issue with our data generation procedure for the graph salad set. We normally require that any target graph chosen for a graph salad satisfy the constraint that all three merge points are mutually reachable from each other via statements from only that component graph/narrative. This is the condition which

determines how many separate instances of the salad we create, each with a different target graph which meets this qualification.

Although the reachability constraint was true for our data set when we created it, we found it necessary to crop our graphs and remove statements in order to reduce excessive file sizes, which means that merge points in a particular salad are not necessarily mutually reachable via statements from the target graph, after all.

(Note that the reachability constraint arose out of a misunderstanding of task desiderata; while we initially thought that it was desirable to have a model capable of connecting initially unconnected seed subgraphs, we later found out that this is not the case; i.e., initially unconnected seed subgraphs should not be connected during narrative expansion.)

Although cropping invalidated the reachability constraint for a portion of our data, given that reachability is a questionable property in the first place, we leverage the fact that our graph salad data set contains both instances which do and do not respect this constraint to determine what effect reachability has, if any, on performance. We discuss our conclusions in **Section 3.6**.

3.3.8 Future Work

Topical Similarity

In the future, it would be worthwhile to explore the use of some measure of topical similarity (as Wang et al. do) when selecting sources to be mixed. Although one might argue that topical similarity is insufficient in this case (as two geopolitically unrelated conflicts, for instance, are likely to heavily share the same sets of generic events [e.g., "bombing," "attack," "retreat,"] and entities ["artillery," "tanks," "soldiers"]), such a measure would allow for a clearer picture of how distinct documents being mixed are.

Statement Variety

Additionally, in **Section 3.3.7**, we remarked that graph salads can become dominated by particular component graphs in regions away from merge points. Such a characteristic might allow for trivial inference; should our model first admit the statement linking "strafing" and "Japanese" in **Figure 3.6**, for instance, it would gain access to an overwhelming number of candidate statements from the target

graph (which, in turn, lead to other statements from the target graph) that would then have a dominant influence during inference. Although this effect is somewhat mitigated when merge points are reachable from other (thereby causing an inference model to encounter more than one branching path), this still poses the risk of making our data set less challenging.

Our initial data generation process required only that entities and events to be merged overlap in ontology label, not in name (hence, "Ulysses S. Grant" and "Mary II" might be matched by virtue of the fact they they are both people). While this relaxed constraint is likely to induce less topical similarity than our name-matching requirement, it might be valuable to use such a constraint to increase statement variety in regions away from named-matched merge points. In **Figure 3.6**, for example, we might increase statement variety around the "Japanese" entity by selecting additional "GeopoliticalEntity" nodes from other sources and injecting their statements around the entity. Because matching on ontology labels alone is considerably less restrictive than matching on both names and ontology labels, this would allow us to introduce (albeit noisier) variety in regions of the graph which are far away from name-matched merge points without imposing overly stringent matching requirements.

3.4 Model for Inference over Graph Salads

Int this section, we discuss the architecture of the model we use to perform inference over graph salads.

3.4.1 Choice of Architecture

Critically, knowledge graphs not only contain information in nodes themselves, but also in the overall layout and context in which those nodes appear. Hence, it is necessary to employ a means of developing representations which encode not only information explicitly belonging to individual nodes, but also information relaying how those nodes are connected to each other in the context of the graph.

Consider, in particular, the task at hand; for our graph salad set, we ask our model to perform an iterative admission procedure in which it adds statements in a KG one-by-one to an ever-growing contiguous narrative. Because our model's decision in a given step dictates which areas of the graph it will have access to in subsequent steps,

it is important that we use an architecture that is capable of telling us how promising a particular statement is in terms of its ability to increase the availability of narratively coherent statements in the future (i.e., "Looking at the context surrounding this statement, is it likely that adding this statement would allow access to an area of the graph which offers statements the model is likely to add in the future?").

To address the issues outlined above, we use a graph convolutional network as the basis for our neural architectures.

3.4.2 Introduction to Graph Convolutional Networks

Graph convolutional networks (Kipf and Welling, 2017) are an increasingly popular means of propagating information throughout a graph and developing node-level representations during training. Graph convolutional networks, or GCNs, were introduced by Kipf and Welling in 2017 as a first-order approximation of spectral graph convolutions (Hammond et al., 2009). GCNs apply convolution-like operations on neighborhoods of nodes; at each layer of a GCN, a node's subsequent embedding is a function of that node's previous embedding and the embeddings of the surrounding nodes. As in traditional convolutional networks (LeCun et al., 1989), successive layers in a GCN produce increasingly abstracted representations of a graph's information and layout.

3.4.3 Conversion to Bipartite Structure

Note that our task requires the model to admit *statements*, rather than EREs, to the narrative-so-far; the EREs are incidental in the sense that any time a statement attached to an ERE is admitted to the growing narrative, the ERE itself also necessarily becomes involved in the story. Hence, we must evaluate the worth of candidate statements based on some explicit representations of the statements themselves. Since statements are represented as edges in the KGs shown in previous figures, we need some way to allow our network to maintain and develop representations of edges (as well as nodes).

To remedy this issue, the first step in our training pipeline is to convert KG instances into a bipartite form in which both EREs **and** statements are represented as nodes.

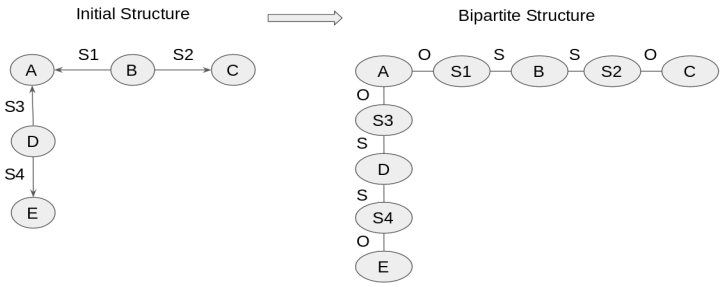

Figure 3.7: Conversion to bipartite structure

Examine **Figure 3.7**. The image on the left represents the structure of a KG produced using our synthetic data generation process. Assume that nodes A-E are EREs, and that edges S1-S4 are statements. The image on the right represents this KG converted into a new form in which both nodes and edges from the original graph are represented using nodes. Note that the resulting graph is bipartite: it satisfies the condition that all nodes can be colored with only two colors in such a way that no node of a particular color is adjacent to another node of that same color.

Consider the statement S1 linking EREs B and A in the original graph. In the transformed graph, S1 is now represented alongside EREs A and B as a node in the graph, with "dummy" edges indicating that A is the object ("O") of S1, and that B is the subject ("S").

With this new structure, we can now develop embeddings for both EREs and statements during training and allow both to inform the admission of new statements into the narrative. Similarly, our network can now determine how promising a particular candidate statement is by operating directly on its explicit embedding. These properties factor importantly into our attention-based inference mechanism in **Section 3.4.5**.

3.4.4 GCN Update Procedure

Note that much of the material in this section is either modeled after or taken verbatim from our publication to the Text Analysis Conference (Cheng et al., 2019), for which the author worked on synthetic data creation and the modeling/training of that work's neural pipeline.

Since we operate on a bipartite graph, and since EREs and statements are in-

herently different with regard to the types of information they represent, we extend conventional GCN update rules to a setting in which EREs and statements are alternatively updated.

The rule for updating the hidden representation h_e of an ERE e is as follows (taken directly from our paper in the 2019 Text Analysis Conference (Cheng et al., 2019)), where our notation is modeled after that in (Marcheggiani and Titov, 2017):

$$\mathbf{h}_e^{l+1} = ReLU\left(\mathbf{W}_{ere}^{l}\mathbf{h}_e^{l} + b_{ere}^{l}\right) + \sum_{s \in N_{stmt}(e)} ReLU\left(\mathbf{W}_{D(e,s)}^{l}\mathbf{h}_s^{l} + b_{D(e,s)}^{l}\right) \tag{3.1}$$

where l is the current layer of the GCN network, $\mathbf{W}^{l}{}_{ere}$ is a self-transformation at layer l for the ERE being updated, $N_{stmt}(e)$ is the set of all statements adjacent to the ERE e, and $\mathbf{W}^{l}{}_{D(e,s)}$ is a linear layer for processing adjacent statements, conditioned on whether the ERE is the subject or the object of each statement (hence, D for "direction").

The (similar) rule for updating the hidden representation h_s of a statement s is as follows:

$$\mathbf{h}_s^{l+1} = ReLU\left(\mathbf{W}_{stmt}^{l}\mathbf{h}_s^{l} + b_{stmt}^{l}\right) + \sum_{e \in N_{ere}(s)} ReLU\left(\mathbf{W}_{D(s,e)}^{l}\mathbf{h}_e^{l+1} + b_{D(s,e)}^{l}\right) \tag{3.2}$$

As shown in **Figure 3.8**, we apply the above two rules in succession for each layer of our GCN. This setup is similar in nature to the "dual-primal" approach detailed in (Monti et al., 2018). Note that, in this new paradigm, a single ERE-to-ERE traversal in a GCN applied to the original format (e.g., in **Figure 3.7**, A-to-B via S1) now corresponds to a total of two ERE-to-statement/statement-to-ERE traversals (e.g., A-to-S1 via dummy edge "O" and S1-to-B via dummy edge "S"). As a result, applying the above two rules in sequence yields the same rate of information flow present in a conventional GCN update rule (i.e., information propagates from A to B in a single layer). We find that a total of two GCN layers yields the best results in our experiments.

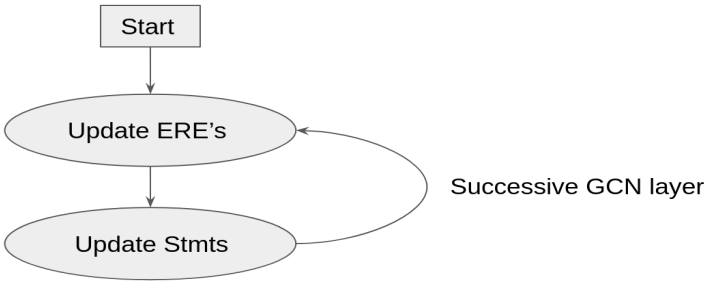

Figure 3.8: GCN update procedure

For a given sequence of statement admissions in a particular graph salad, the GCN embeddings for that KG are computed only once (namely, before the model admits any statements). Note that only the set of query statements changes from step to step in iterative admission; because losses are computed only after the model is done admitting statements entirely, there is no reason to continue updating our GCN embeddings throughout the sequence of admissions.

3.4.5 Attention-based Inference

Once embeddings have been computed for the EREs and statements in a KG, the model must decide on a one-by-one basis which available statements to add to the narrative-so-far. At any given step in the sequence of admissions, the set of statements which are attached to any ERE captured by the narrative-so-far (query) set, but have not yet been added to the narrative-so-far, is the set of *candidate statements*. Said another way, candidate statements are statements on the fringe of the narrative-so-far at a given time.

In order to determine how well candidate statements cohere with the narrative-so-far, we employ attention mechanisms like those in (Wang et al., 2018) and (Wang et al., 2019) which develop context vectors to supplement the candidate statement representations. As those works compute attention scores between possible candidate sentences and sentences in a growing narrative, for each candidate statement (the "attender"), we likewise compute an attention score between its embedding and the embeddings of the EREs and statements comprising the query set (the "attendees"). Since ERE and statement embeddings represent different types of information, we employ two separate attention networks for score calculation.

We experiment with two forms of attention from (Luong et al., 2015): (a) bilinear attention and (b) concatenative attention, defined as

$$score_{bilinear}(\mathbf{h}_{s_c}, \mathbf{h}_{s_i}) = \mathbf{h}_{s_c}^T \mathbf{W}_{sts} \mathbf{h}_{s_i} \tag{3.3}$$

$$score_{concat}(\mathbf{h}_{s_c}, \mathbf{h}_{s_i}) = \mathbf{v}_{sts}^T tanh(\mathbf{W}_{sts}[\mathbf{h}_{s_c}; \mathbf{h}_{s_i}]) \tag{3.4}$$

where $\mathbf{h}_{s_c}$ and $\mathbf{h}_{s_i}$ are the final GCN embeddings for candidate statement s_c and query statement s_i, respectively; $\mathbf{W}_{sts}$ is a linear transformation used when calculating statement-to-statement attention scores; and $\mathbf{v}_{sts}$ is an additional linear layer. For calculating scores between candidate statements and hypothesis EREs, we reuse the above equations, swapping out $\mathbf{W}_{sts}$ and $\mathbf{v}_{sts}$ for a different set of neural layers $\mathbf{W}_{ste}$ and $\mathbf{v}_{ste}$ (for "statement-to-ERE").

After obtaining attention scores, we then apply the softmax function to obtain a weight for each pairing of candidate statement s_c and query statement s_i (also from Luong et al., 2015):

$$\mathbf{a}_{s_c}(s_i) = \frac{exp(score(\mathbf{h}_{s_c}, \mathbf{h}_{s_i}))}{\sum_{i'} exp(score(\mathbf{h}_{s_c}, \mathbf{h}_{s_{i'}}))} \tag{3.5}$$

Lastly, for each candidate statement, we obtain a weighted sum (context vector) of query statement representations:

$$\mathbf{r}_{sts}(s_c) = \sum_{i'} \mathbf{a}_{s_c}(s_i) \mathbf{h}_{s_i} \tag{3.6}$$

See **Figure 3.9** for an illustration of this attention procedure, where red arrows indicate statement-to-ERE attention, and blue arrows statement-to-statement attention.

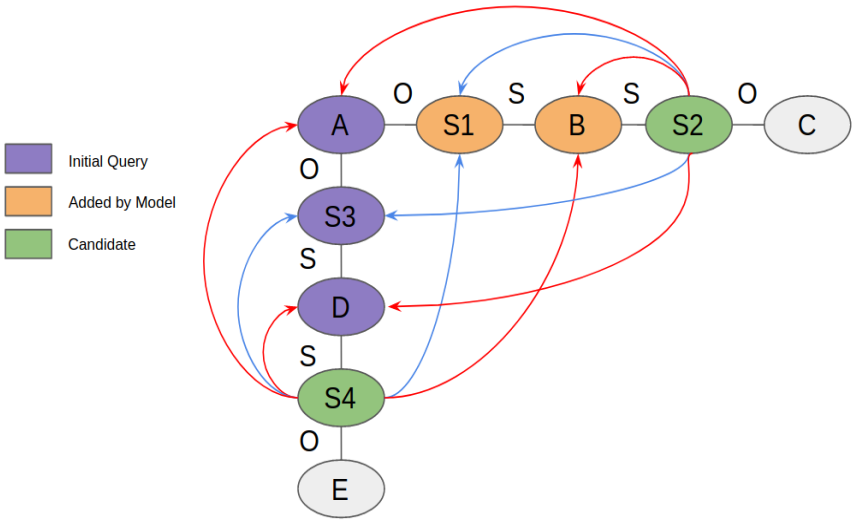

Figure 3.9: Illustration of attention-based inference

As earlier, we repeat this process for each pairing of candidate statement s_c and query ERE e_j via a second neural network to obtain a weighted sum $\mathbf{r}_{ste}(s_c)$ of query ERE representations for each candidate statement.

We then concatenate each candidate statement's final GCN embedding h_{s_c} with its statement-to-statement and statement-to-ERE context vectors $\mathbf{r}_{sts}(s_c)$ and $\mathbf{r}_{ste}(s_c)$ and run this through another neural layer and the `tanh` nonlinearity.

Lastly, we feed the results from the tanh layer above into a final layer with a single output and apply the `softmax` function over all candidate statements to obtain a coherence ranking for each candidate. Our model admits the candidate statement with the highest ranking to the narrative-so-far.

A diagram illustrating the neural pipeline for a given admission step of the model we use on graph salads is shown in **Figure 3.10**. We refer to this model as M_{SALAD} (for "graph salads").

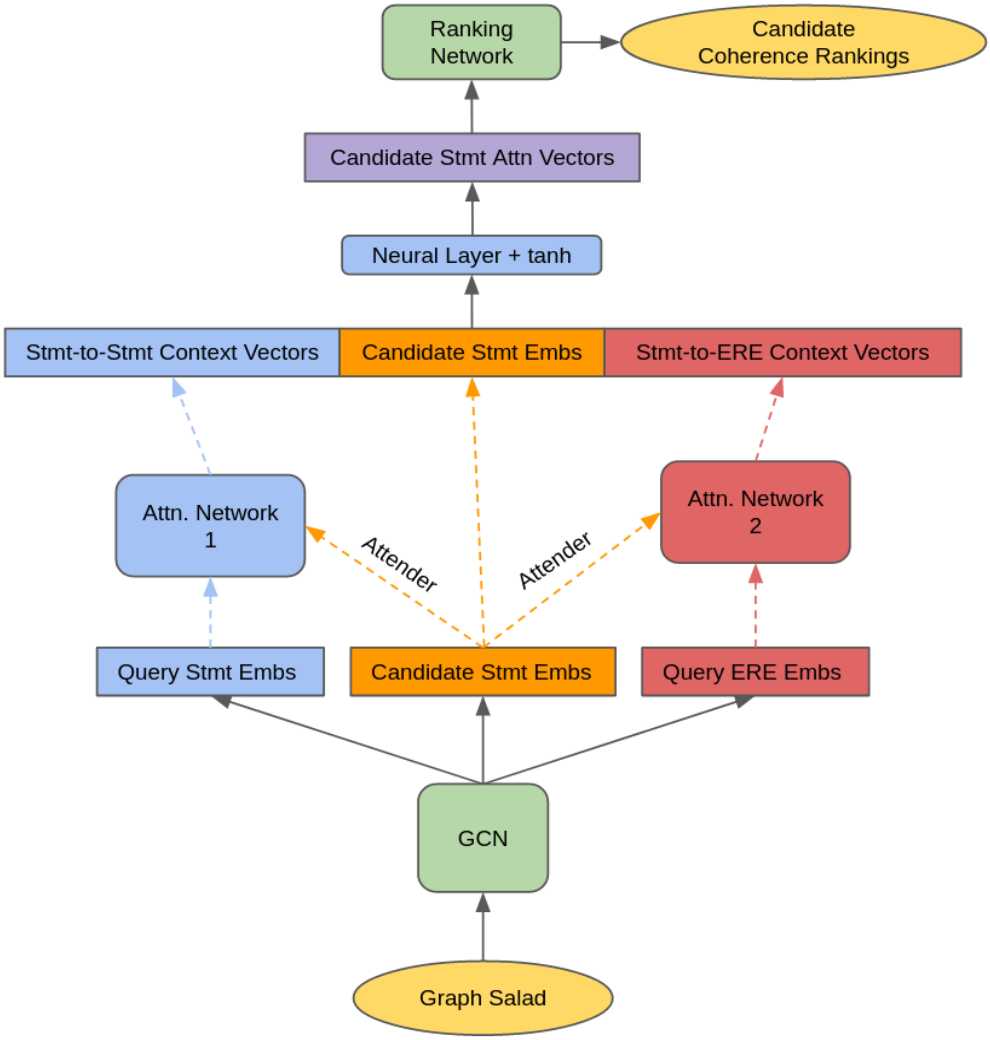

Figure 3.10: Pipeline for graph salad model (M_{SALAD})

3.5 Experiments on Graph Salads (M_{SALAD})

Our data set for our M_{SALAD} experiments consists of a training set of 60k graph salads and a test set of 10k graph salads. Model validation on a held-out set of 2.5k mixtures occurs every 15k training instances. So as to appropriately hold out data, no source document used to create a mixture in any one of the three data set types is used in the other two. For a given graph salad, we ask the model to admit statements to the provided seed subgraph for that instance until one of the following conditions is met:

- The model has admitted a total of 25 statements
- The number of statements the model has admitted is equal to the number of target-narrative statements that could have been added (i.e., we do not want to ask the model to admit 25 statements when there are only 12 target-narrative statements to admit)

This is analogous to the "fixed number of sentences" constraint used in the one-class clustering task in (Wang et al., 2019).

We maximize the marginal likelihood of admitting any target-graph candidate statement. Our performance metric is precision over admitted statements (i.e., what percentage of statements the model chose to admit were actually from the target narrative?). We initialize all name and ontology token embeddings with pretrained 300-dimensional vectors from a Word2Vec model (Mikolov et al., 2013a;b;c) trained on a portion of the GoogleNews corpus at `https://code.google.com/archive/p/word2vec/`.

As discussed in **Section 3.4.5**, we experiment with both concatenative and bilinear attention after (Luong et al., 2015). Since the task involving our graph salad set is one of iterative statement admission, a poor set of decisions early on in a sequence of admissions can derail a model's ability to learn. As a result, we employ a form of teacher forcing whereby, at each admission step, we admit with some probability a candidate statement from the target narrative if the model's proposed choice is not from the target narrative. We initialize the probability of teacher forcing on a given admission step to 0.25 and anneal it after every 3k graph salads by a factor of 0.9659.

Recall that, at a given admission step, there are generally multiple target-narrative candidate statements on the fringe of the narrative-so-far subgraph. As a result, we experiment with the following two settings of teacher forcing:

- Admit a random candidate statement from the target narrative
- Admit the gold candidate statement from the target narrative which the model ranks most highly among current candidates

Lastly, we try a run in which the model is given not only query statements surrounding the single most highly connected merge point, but query statements around

all merge points in the salad; this is intended to determine the effect of the size and spatial distribution of the query set on performance.

We apply dropout to GCN layers and attention layers with a probability of 0.5 and 0.3, respectively. We use a learning rate of 5e-5. We use the Adam optimizer (Kingma and Ba, 2015) with a weight decay of .001 for this set of experiments. A full list of hyperparameter values can be found in **Section A.1** in the appendices.

3.6 Results

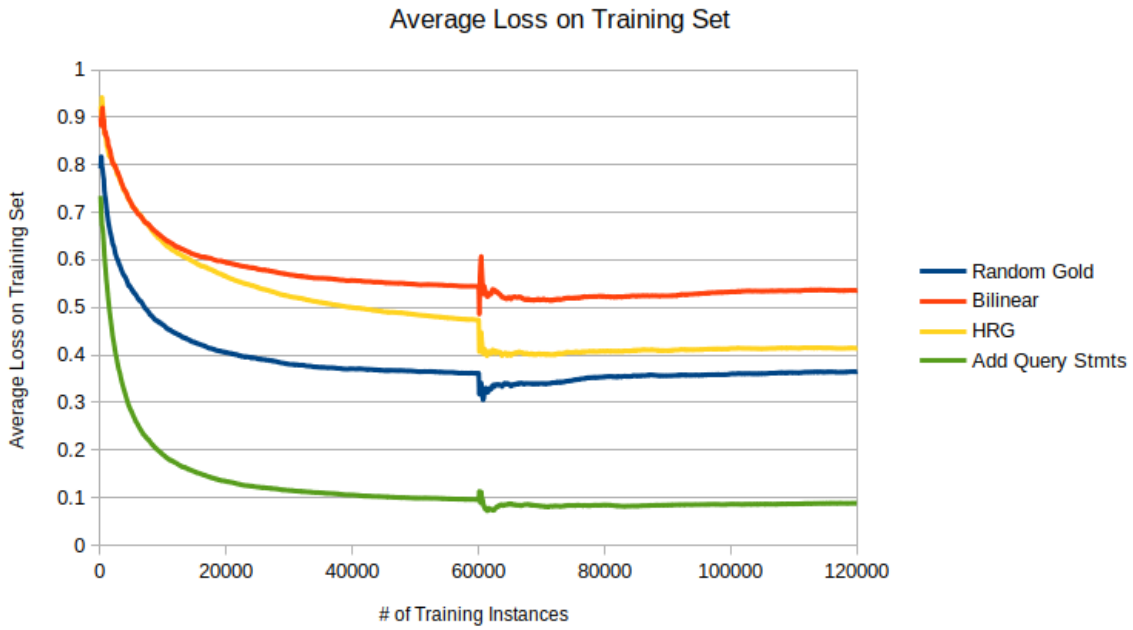

Figure 3.11: Average loss on graph salad training set for M_{SALAD}, where:
Random Gold = random gold when teacher forcing
Bilinear = bilinear-style attention
HRG = highest ranking gold when teacher forcing
Additional Query Statements = query statements around *all* merge points were given, rather than around only the most connected merge point

	Random Gold	HRG	Bilinear	Add Query Stmts
Entire Test Set	.90	.91	.77	.97
Reach Subset	.90	.86	.79	.98
Non-reach Subset	.79	.79	.74	.96

Table 3.2: Average precision on graph salad test set for M_{SALAD}, where:
Reach Subset = subset of graph salad instances with at least three merge points which are mutually reachable via target-graph statements
Non-reach subset = subset of graph salad instances without reachability guarantees
Bilinear = bilinear-style attention
HRG = highest ranking gold when teacher forcing
Additional Query Statements = query statements around *all* merge points were given, rather than around only the most connected merge point

We display training curves for our four variants of the M_{SALAD} model in **Figure 3.11**. The first variant in the legend corresponds to the model which uses concatenative-style attention, and the second bilinear attention. The third variant admits the highest-ranked gold ("HRG") candidate rather than a random gold candidate when teacher forcing. The fourth variant ("additional query statements") is an experiment in which we add two statements from *all* merge points in a given salad to the initial query set instead of selecting two statements from only the most highly connected merge point.

We additionally isolate (i) the subset of test instances which feature at least three merge points which are mutually reachable from each other via statements from only the target graph and (ii) the subset of test instances which do not satisfy this condition. We theorize that this can have an effect on performance, as merge points present more varied narrative choices to a model, and (therefore) seem more likely to induce challenge during inference. The results for these tests are displayed in **Table 3.2**, with the "Entire Test Set" column representing scores over all test instances.

Clearly, concatenative attention is more suited to our task than bilinear attention. The model which admits the highest-ranking gold when teacher-forcing sees performance that is on par with that of the default teacher-forcing model; although this model sees a decrease in precision on the subset of instances with mutually reachable merge points, it appears that the difference between using the highest ranked gold vs. a random candidate is negligible.

With precision scores of around .97, the model which is given access to query statements around all merge points clearly faces a trivial task; it seems that any potential increases in challenge due to more immediate exposure to more varied statements around each merge point are offset by the fact that the model starts with more narrative direction.

3.7 Reinforcement Learning Extension

As an extension to our M_{SALAD} model, we additionally explore framing our task in a reinforcement learning setting for reasons we explain below. This section details the setup for this environment and the associated architectural changes.

3.7.1 Motivation for Reinforcement Learning

In this section, we motivate and review our exploration of proximal policy optimization, a state-of-the-art reinforcement learning algorithm, for use in our task. As a quick review, in a reinforcement learning setting, an *agent* with some set of tunable parameters learns how to interact with its *environment* in (typically) discrete time steps. At each time step, the agent attempts to determine which possible *action* it can take will lead to the highest future *return*, a discounted sum of expected *rewards* in future steps.

Longer-Range Contextualization

A GCN's inability to scale well with a large number of neural layers is a clear limitation of the paradigm (Marcheggiani and Titov, 2017, Chen et al., 2019). In our particular task, this means our model is incapable of picking up on longer-range contextual associations within a KG; information can propagate over at most two statement traversals from a given statement.

RL has the capability to forecast and model long sequences of decision-making. Monte Carlo rollouts, for instance, allow an agent to simulate sequences of actions and propagate more distant rewards back to earlier nodes.

Reward Shaping

A similar motivation for framing our task in an RL setting is the fine-grained reward shaping capability afforded by RL. RL algorithms present the opportunity to tune reward functions and incorporate more fine-grained desiderata during the training process itself. These properties might allow us to more easily enforce desired constraints; we might, for instance, assign a highly negative reward to the admission of a statement which belongs to an impossible scenario (e.g., a sniper assassinating himself). Along similar lines, we might encourage the model to explore more diverse regions of a graph by diminishing the reward an agent receives when focusing too heavily on a small subset of EREs in contiguous time steps.

3.7.2 RL Components

In this section, we outline how various pieces of an RL training paradigm (states, actions, and rewards) are represented in our particular task.

First, we represent a state at any particular time step of a graph salad admission sequence by (i) a list of the GCN embeddings for the query set and (ii) a list of the GCN embeddings for the candidate statement set. An action at a particular time step corresponds to the admission of a statement from the candidate set into the query set. Lastly, we provide the agent with a reward of +1 if a given candidate statement it chooses to admit is from the target component graph, and a -2 otherwise.

A "trajectory" or "episode" in our setting is a sequence of admissions; the "starting" state of a trajectory represents the case when no statements have yet been admitted and the model is presented with the initial set of query statements. A trajectory terminates when the designated number of statement admissions for a particular graph instance is reached.

3.7.3 Algorithm: Proximal Policy Optimization

The implementation ideas presented in **Section 3.7.1** represent end-goal hopes for mechanisms which RL might eventually allow us to incorporate into our training process; before we even attempt these ideas, we must first frame our task in a more basic RL setting to prove the viability of an RL algorithm in our task.

For this purpose, we turn to a SoTA policy-gradient method which fits nicely into our existing paradigm. Proximal policy optimization (Schulman et al., 2017) is an actor-critic-style method intended to address the robustness and efficiency issues of prior deep RL approaches. In actor-critic algorithms (Konda and Tsitsiklis, 2000), both a policy network (the "actor") and a value function network (the "critic") are trained. The actor decides which action to take given the current state, and the critic judges how good taking that action in that state is.

In classical policy gradient methods (Williams, 1992, Sutton et al., 2000), and (thus) in PPO/actor-critic methods, the actor policy is typically represented by the following, after (Sutton and Barto, 2018):

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{3.7}$$

, where a is an action in state s, and h is a preference for action a in state s given the parameters θ. Hence, an actor (in this case) forms a probabilistic distribution over actions in a given state by `softmax`-ing over preferences for actions in a state. In our case, our actor network is simply the network from **Section 3.4** which forms a probabilistic distribution over candidate statements by computing context vectors for the candidates.

In actor-critic methods, a value function (the "critic") is used in conjunction with an actor to reduce variance. A value function $V_\phi(s)$ takes as input the current state and produces a scalar value estimating the expected future return of being in that state, given some parameterization ϕ of the value function. For our value function, we use a network which takes as input the attention vectors for each candidate statement at a particular time step and outputs a scalar value. To estimate the value of a given state, we take the mean of all the scalars produced by the network (hence, the estimated value is an average of an estimation of each of the actions' values in that particular state).

The fundamental principle behind PPO is to limit the degree to which the actor network's parameters can change during an update. Schulman et al. present both an approach that uses a KL penalty and one that uses clipped ratios; we follow the latter. Let our policy network parameterized by θ be π_θ; let the current time step be t; and let the state s_t, the action taken a_t, and the advantage function $\hat{A}_\mathrm{t}$ (defined

below) be with respect to time step t.

Assume we collect a set of (possibly partial) trajectories under the current policy. Under PPO, we first compute the advantage estimates at each time step via the recursive rule for generalized advantage estimation (Schulman et al., 2015), with notation following that found in (Schulman et al., 2017):

$$\begin{aligned} \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ with \qquad \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{3.8}$$

where V is our function approximator, γ is the discount factor, λ is the trace decay rate, and T is the horizon. An advantage at a given time step represents a measure of how far off our value function's estimation of a state's value is from a more empirically accurate estimation (in this case, a weighted sum of future rewards).

Schulman, et al. then define a ratio $r_t(\theta)$ thusly:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{3.9}$$

where $\pi_{\theta_{old}}$ is a behavior policy used to collect experiences.

Now, assume we collect a set of trajectories under a policy $\pi_{\theta_{old}}$. Schulman et al. then perform a series of gradient steps on the objective using the Adam optimizer (Kingma and Ba, 2015):

$$L_t^{CLIP}(\theta) = \hat{E}_t\Big[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{3.10}$$

where *clip* means that the ratio $r_t(\theta)$ can go no lower than 1 - ϵ, and no higher than $1 + \epsilon$.

A series of gradient descent steps are then performed on a separate loss function *assuming the actor and critic do not share learnable parameters, which is the approach we take (notation after Schulman et al., 2017):

$$L_t^{VF}(\phi) = (V_\phi(s_t) - (\hat{A}_t + V_{\phi_{old}}(s_t)))^2 \tag{3.11}$$

where $V_{\phi_{old}}$ is the value function approximator at the time the advantage estimates were computed (i.e., right after the experiences in D were generated) and V_ϕ is the

current critic network.

We similarly experiment with a clipped value function (objective following that in (Jayasiri, 2019)):

$$L_t^{VF}(\phi) = \frac{1}{2}\hat{E}_t\Big[max((V_\phi(s_t) - R_t)^2, (V_{\phi_{CLIP}}(s_t) - R_t)^2)\Big] \tag{3.12}$$

where

$$V_{\phi_{CLIP}}(s_t) = clip(V_\phi(s_t) - \hat{V}_t, -\epsilon, +\epsilon) \tag{3.13}$$

R_t here is the discounted expected future return at time t, or $\hat{A}_t + \hat{V}_t$; $\hat{V}_t$ is the state value at time t at the time the advantage values were calculated.

We follow OpenAI's Spinning Up implementation (at `https://spinningup.openai.com/en/latest/algorithms/ppo.html`) and take the mean loss over all time steps and all sample trajectories in a given batch of experiences during either gradient descent step.

Hence, for each iteration of the algorithm, we generate trajectories of length T by having our actor admit T statements to the initial narrative subgraph, we calculate the advantages at each time step for each trajectory in the batch, and we then perform some number of gradient descent steps on the above objectives before generating a new set of experiences. Note that the ratio $r_t(\theta)$ is recalculated using the new π_θ obtained after each successive step of gradient descent (and the value objective with the new V_ϕ).

3.7.4 Model Architecture for RL

Given the complexity of learning an end-to-end RL model from scratch, we first pretrain an actor whose architecture is the same as that presented for use with graph salads in **Section 3.4**. The actor is pretrained on a held-out subset of graph salads, and the remaining salads are reserved for training the critic network. The graph convolutional network of the actor alone is frozen before we begin training the critic; the actor's attention network continues to learn after the pretraining stage.

The critic is a module which takes as input the set of candidate attention vectors and produces a scalar estimating the value of being in the state which allows that particular configuration of candidates. We experiment with two cases. In case 1

(the "single" case), the critic module is a simple additional linear neural layer which transforms the candidate attention vectors into scalars and returns the mean of the resulting logits as the value of the current state; in this setup, the critic simply uses the attention network of the actor. In case 2 (the "dual" case), the critic module is assigned a wholly separate attention network from the actor, similarly converting candidate attention vectors into scalars and returning the mean of the resulting logits.

An illustration of how the RL modules fit into the context of the architecture in **Figure 3.10** is depicted in **Figure 3.12**. (Note that the case depicted is the "single case," where the critic shares the attention network of the actor). We refer to our RL-augmented model as $M_{SALAD\text{-}RL}$

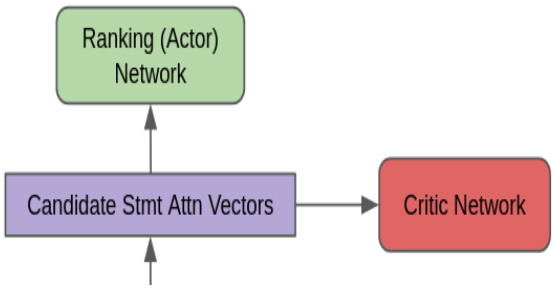

Figure 3.12: Pipeline for RL modules

3.8 Experiments on Graph Salads with Reinforcement Learning Extension ($M_{SALAD\text{-}RL}$)

(Note: the experiments presented here for our RL setting were performed on an earlier iteration of our graph salad data set which, unlike the set on which we reported results in **Section 3.6**, did not require name-matching among merged nodes. Hence, the experimental results from **Section 3.6** cannot be directly compared to the ones presented here. We do, however, apply our M_{SALAD} model to the data set in these experiments as a baseline.)

We first pretrain (or "warm-start") an M_{SALAD} actor network in our non-RL supervised setting for two epochs on a 20k subset of 80k training graph salads. We use a learning rate of 1e-5 and dropout rates of .5 and .2 for the convolution and attention networks, respectively. We start teacher forcing at a probability of .95 and anneal it over time to a final value of .5 at the end of pretraining. For our baseline,

we take our pretrained actor and simply continue to train in a supervised fashion on the 60k remaining training graph salads for a total of 2 epochs, resetting the initial probability of teacher forcing to .8 and annealing it over time. We continue with the learning rate and dropout rates used during pretraining.

For our RL-based models, we position both the pretrained actor and an additional "critic" network in a PPO-based RL setting. We set the maximum trajectory length of generated samples to be 25, the number previously used as the maximum number of statement admissions in **Section 3.5**. Although the maximum length is 25, in practice, we stop all actors in a batch of trajectories once any one of them has reached the maximum number of extractions defined for itself. We note that this is a simplifying assumption made for the sake of ease when dealing with graph batching for our RL experiments.

We train our critic and pretrained actor in our RL setting for two episodes per training instance (hence, a total of 120k episodes). We try both a clipped (with ϵ=.2) and unclipped value function and learning rates of 1e-5 and 1e-4 for both our single and dual attention networks. Additional RL-specific parameters can be found in **Section A.2** in the appendices.

3.9 Results on RL Experiments

(Note: we present a test precision score, but not training curves, for our baseline, as our baseline's average precision on the training set cannot be directly compared to those of the RL-based variants; this is because our supervised baseline uses teacher forcing, but our RL-based methods rely solely on the decisions of the actor.)

3.9.1 Single Attention Network

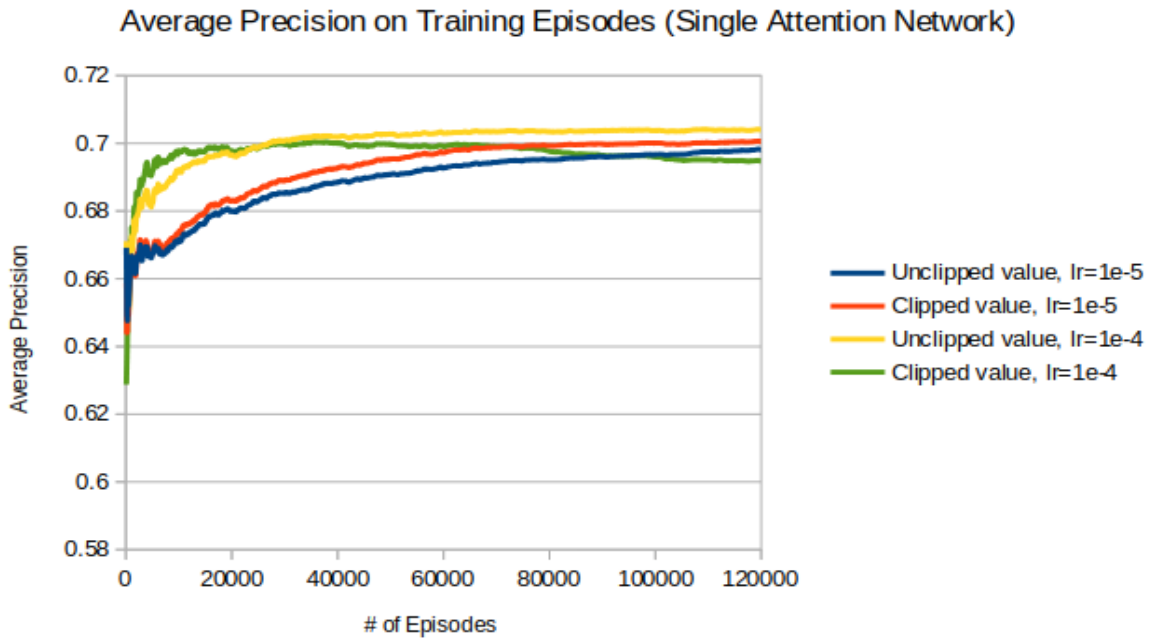

Figure 3.13: Training curves (average precision) for "single" attention variants of $\text{M}_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

In **Figure 3.13**, we present the training curves (in the form of average precision) for our variant with a single attention network shared by the actor and critic. **Note that the scale of the y-axis begins at a precision score of .58 and ends at a maximum of .72–thus, any changes in performance here are quite small.** When observed in conjunction with the results of these runs on our test set in **Table 3.3**, we see virtually no statistically significant differences between any of the four variants tested.

	Single (Fixed) Attention Network
Unclipped VF, lr=1e-5	.6602
Clipped VF, lr=1e-5	.6607
Unclipped VF, lr=1e-4	.6621
Clipped VF, lr=1e-4	.6603

Table 3.3: Average precision on test set for "single" attention variants of $\text{M}_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

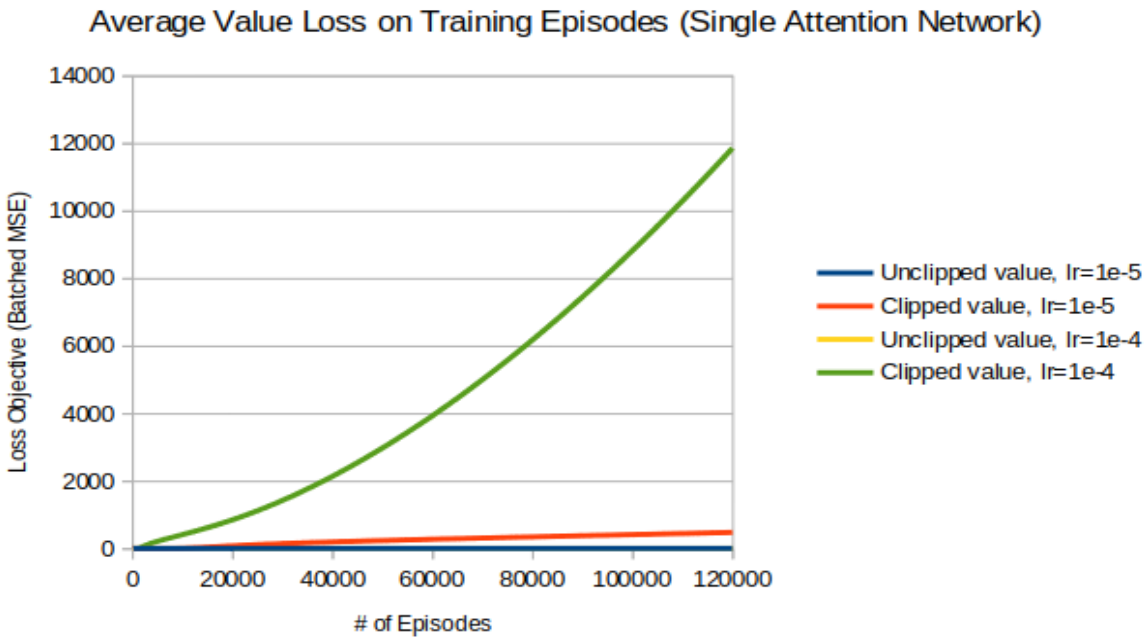

Figure 3.14: Training curves (average loss) for "single" attention variants of $\text{M}_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

We display the average loss on training episodes for our single attention network variants in **Figure 3.14**; we note that using clipping in conjunction with a larger learning rate appears to cause the value function loss to explode.

3.9.2 Dual Attention Networks

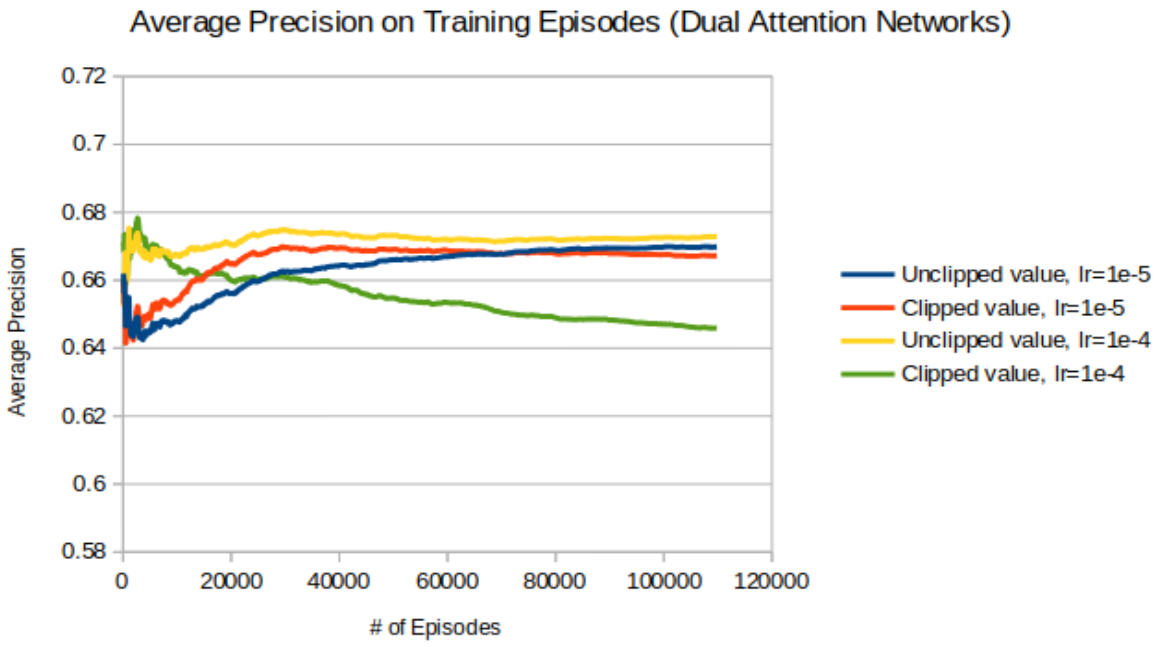

Figure 3.15: Training curves (average precision) for "dual" attention variants of $\text{M}_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

Likewise, we display average precision per training episode for our dual setup in **Figure 3.15**. **Again, note the scale of the y-axis here.** Average precision scores on the test set are (again) displayed in **Table 3.4**. Results here appear similarly underwhelming; there is (again) no indication that our RL-based extension has a visible effect on learning.

	Dual Attention Networks
Unclipped VF, lr=1e-5	.6602
Clipped VF, lr=1e-5	.6605
Unclipped VF, lr=1e-4	.6635
Clipped VF, lr=1e-4	.6577

Table 3.4: Average precision on test set for "dual" attention variants of $M_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

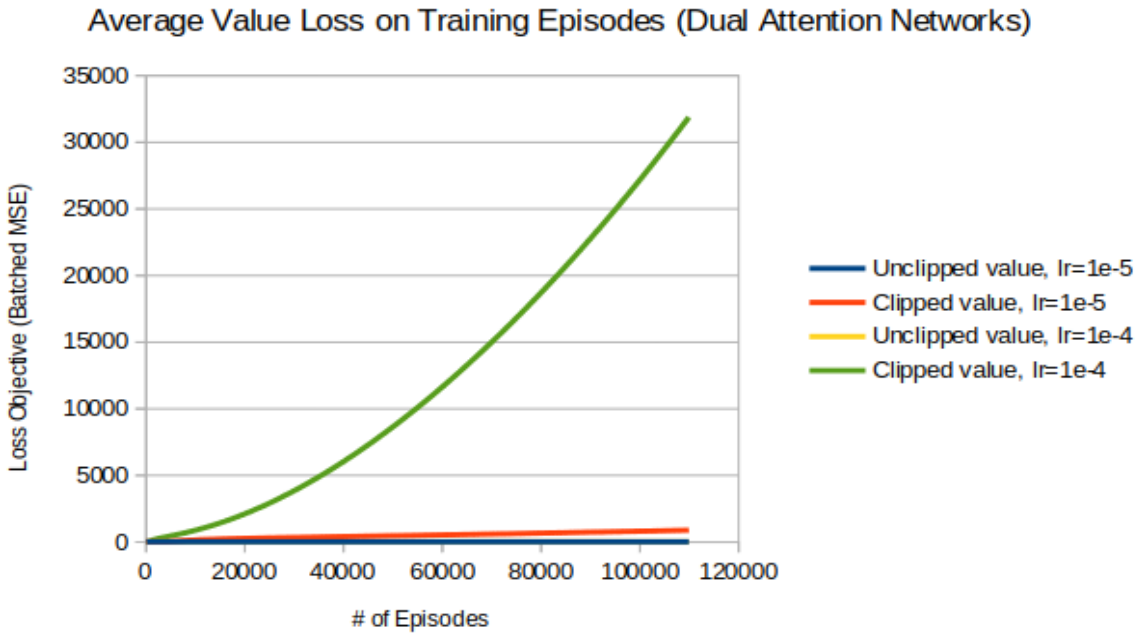

Figure 3.16: Training curves (average loss) for "dual" attention variants of $M_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

An evaluation of the loss curves (shown in **Figure 3.16**) similarly reveals an explosion in value function loss for our clipped configuration with a learning rate of 1e-4.

	Single Attention Network	Dual Attention Networks
Unclipped VF, lr=1e-5	.6602	.6602
Clipped VF, lr=1e-5	.6607	.6605
Unclipped VF, lr=1e-4	.6621	.6635
Clipped VF, lr=1e-4	.6603	.6577

Table 3.5: Average precision on test set for variants of $M_{\text{SALAD-RL}}$, where:
Unclipped value = Unclipped value function
Clipped value = Clipped value function (with $\epsilon = 0.2$)
lr = learning rate

Lastly, we present all test set results for our eight configurations in **Table 3.5**. Our purely supervised baseline achieved an average precision of 0.6550 on our test set, which is comparable to the score of all RL-based variants.

3.9.3 Discussion

Clearly, our RL paradigm failed to have any meaningful impact on learning. A recent case study (Engstrom et al., 2020) suggests that the success of the algorithm we used (PPO) is heavily dependent on a series of code-level implementation details which we were unaware of at the time we conducted the experiments. Given the apparent sensitivity of PPO to these details, and given that would be time-intensive to work our paradigm into existing implementations of PPO on Github, we have tabled RL for now, and offer an alternative avenue through which to achieve longer-range contextualization in our **Future Works** chapter.

Chapter 4

Data Set 2: Cloze-Style Data for use with Contextualized Embeddings

Contextualized embeddings (Peters et al., 2018a, Devlin et al., 2019, Radford et al., 2019) have risen in popularity in recent years as an information-rich evolution of more classical, purely distributional word embeddings (Mikolov et al., 2013a;b;c). A number of works have observed substantial increases in performance over previous state-of-the-art models on various tasks when simply fine-tuning large-scale transformers that are pretrained on language modeling tasks (Peters et al., 2018a, Devlin et al., 2019, Radford et al., 2019). Given these results, it is clear that embeddings produced by large-scale transformers are an accessible way to enrich representations.

In this chapter, we motivate our interest in injecting contextualized embeddings into our task, detail why simply applying contextualized embeddings to our graph salad task would produce a setting in which contextual information is trivially exploitable, and create and work with a set of what we refer to as "cloze-style" data which is intended to address this issue.

4.1 Data Generation

4.1.1 Motivation for Use of Contextualized Embeddings

The difficulty of information extraction necessarily means that the source graphs derived from single documents are often noisy. Some instances of noisy parses are as follows:

- Argument swapping (e.g., a son and his mother give an inheritance to their deceased father; a candidate for public office donates campaign money to a supporter)
- Unspecific events (e.g., a conflict event whose only argument is an attacker; a transaction event in which both parties are specified as the giver, but neither as the recipient)

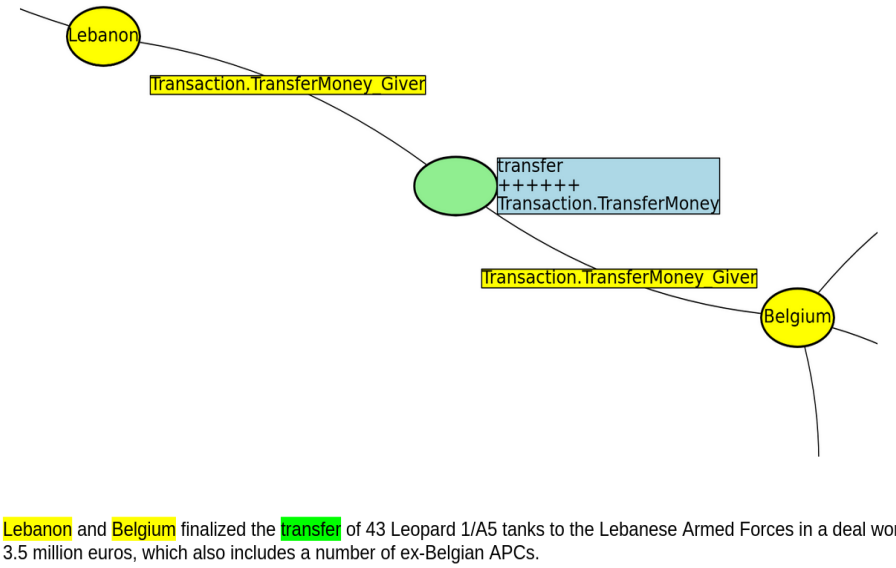

Figure 4.1: Example of unspecific event

- Excessive argument slotting (e.g., a monogamous marriage event with more than two spouses)
- Noisy coreference (e.g., two distinct attack events from a single document being resolved into one; the entity mentions "Revolutionary Nicaragua," "arms," and "women" being resolved into a single entity)

Given that contextualized embeddings take into account the particular textual context in which words appear, we hope that using them in place of more classical models like Word2Vec might allow us to leverage information we are otherwise lacking when faced with problematic parses that do not properly specify how an event played out or which entities were involved.

TA1/TA2 groups are already actively using contextualized embeddings in their pipeline, so it is possible these might be passed on to us (if in a constrained way) in the future.

4.1.2 Transformer Selection

Most TA1/TA2 groups currently use BERT embeddings (Devlin et al., 2019) in their pipelines. Unfortunately, BERT is trained on Wikipedia data, and our synthetic

data is generated using Wikipedia articles. As a result, we cannot guarantee that a model which utilizes BERT embeddings would not have already "seen" the texts used to generate our test data set. Hence, using BERT as a testbed for the viability of contextualized embeddings in our task would prevent us from reserving a subset of unseen data for test purposes.

GPT-2 (Radford et al., 2019), while explicitly avoiding pretraining on Wikipedia data, is unidirectional, which means the arguments appearing later in an event's textual span would observe more context than arguments that appear earlier. As a result of these complications, we decide to use embeddings generated by ELMo (Peters et al., 2018a), which is bidirectional, and which is not trained on Wikipedia data.

4.1.3 Contextual Bleed-over

When exploring the injection of contextualized word embeddings into our task, we initially considered the possibility of simply using them when performing inference on the graph salads data set described in **Chapter 3**. However, an issue (which we describe below) quickly motivated the generation of an entirely new data set for the use of contextualized word embeddings in our setting.

Consider **Figure 4.2**, an illustration of a subgraph taken from one of our graph salads. Below the subgraph is the original sentence from which the event and its arguments were pulled, with each event/entity mention identified by the parser highlighted in the text. Assume that (as depicted in red) the model already knows that "ship" and "Vieques" are target-graph arguments of the deployment event. Now assume the model is asked whether or not Jacksonville and Guantanamo Bay should be considered arguments of the event given the narrative-so-far.

An intuitive way of determining a context for the "deployed" event is to take as a context window the span of text from the earliest mention of any argument to the latest. Hence, in this case, we might define the context window as the piece of text starting at "ship" and ending at "Vieques." Assume we then contextualize each statement's embedding within the context window; since the embeddings for the statements connected to "ship" and "Vieques" are part of the query narrative (and, therefore, inform the admission of new statements at inference time), and since both statements were contextualized on a window which contained "Jacksonville" and

"Guantanamo Bay," the model would be privy to distributional information suggesting "Jacksonville" and "Guantanamo Bay" are arguments to the "deployed" event. Hence, the process of "filling out" the other arguments from a single-source text would be almost trivial. We refer to this issue as "contextual bleed-over."

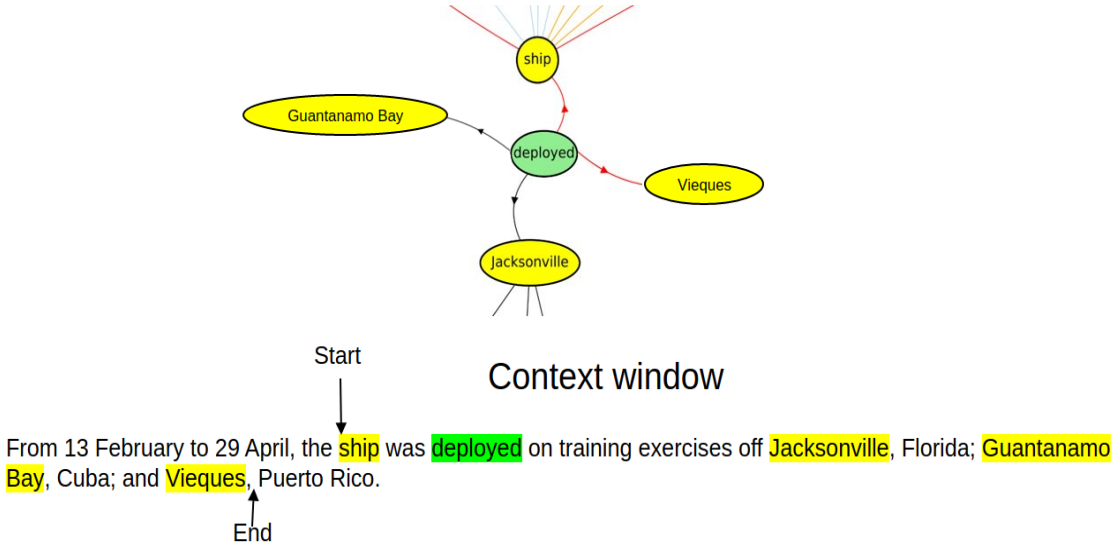

Figure 4.2: Bleed-over issue

4.1.4 Cloze-Style Data Generation

Given the issue outlined above, it is clear that the iterative admission process in our "graph salads" setting would be problematic in the sense that the statements a model admits to a narrative-so-far would trivially reveal which other statements belong to the target narrative by way of their shared context. As a result, we design an alternative data set (the "cloze-style" set) and task which fulfill criteria that ensure the model is unable to leverage contextual bleed-over when extracting statements. We first describe the knowledge graph structure for our cloze-style data, and then describe and motivate the design constraints we impose during data generation.

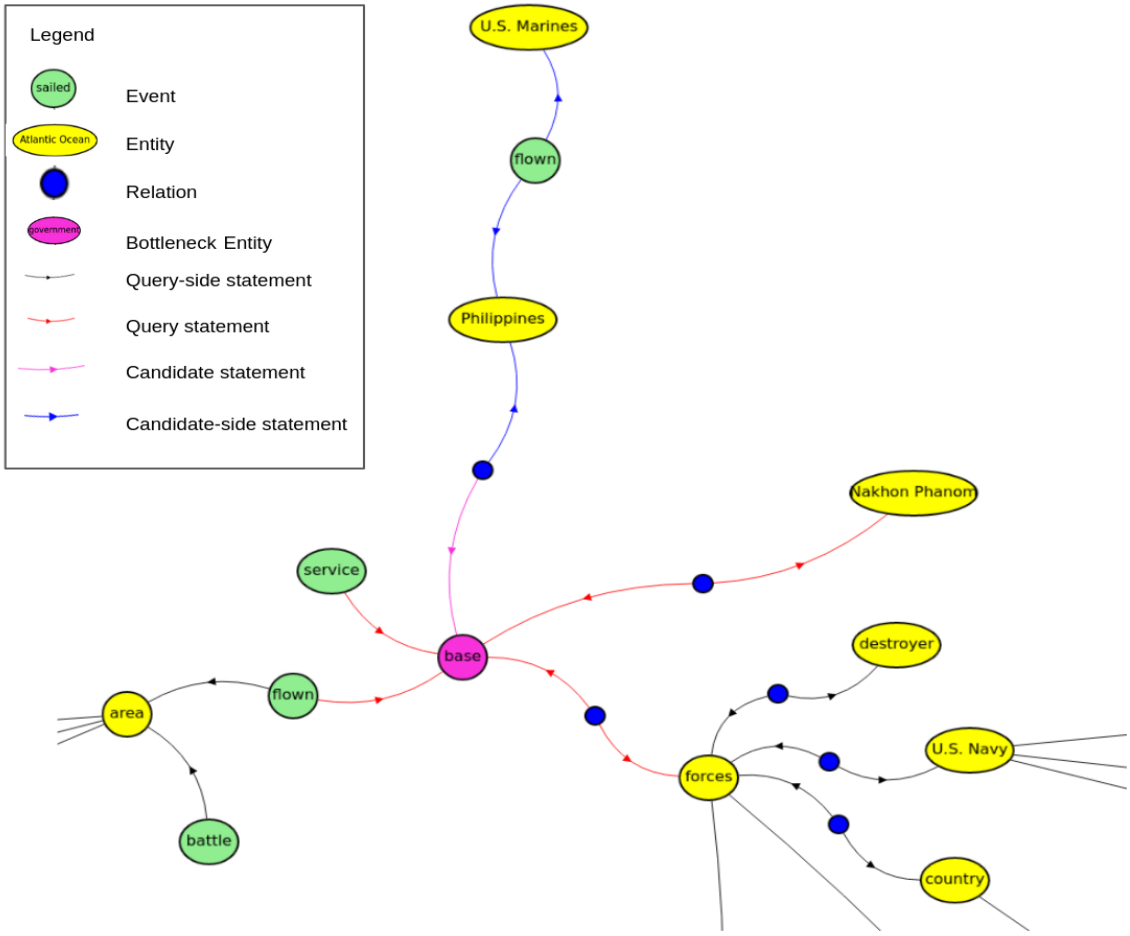

Figure 4.3: Bifurcated structure

Bifurcated Knowledge Graph Structure

In our cloze-style set, we impose a point of bifurcation at a "bottleneck" entity which divides a "query" side and "candidate" side; the "query" side represents a subgraph which contains query statements (statements which are known to be part of the gold narrative), while the "candidate" side represents a subgraph whose pieces are all not currently part of story-so-far/query set. The model is asked whether or not it should admit the statement (namely, the "candidate" statement) linking the "query" and "candidate" sides to the story-so-far.

In **Figure 4.3**, for instance, the entity labeled "base" is a "bottleneck" entity. The "candidate" side statements are colored in blue, while the "query" side statements are

colored in black. The purple statement (linking "base" and "Philippines" via a relation node) is the single candidate statement the model will be asked to admit/reject, and the red statements are the seed/query statements given to the model.

Class Labels

We divide training instances in our cloze-style set into "positive" and "negative" classes; "positive" instances are knowledge graphs whose statements/EREs are derived from only a single source Wikipedia document, while "negative" instances are knowledge graphs derived from two source Wikipedia documents. The instance in **Figure 4.3** contains only statements and EREs drawn from a single source Wikipedia document, and is (therefore) a positive instance: an ideal model should admit the candidate statement to the narrative.

When constructing negative instances, we artificially merge subgraphs from two component sources at the bottleneck entity; as in **Section 3.3.4**, we require that merged entity nodes overlap in ontology type and at least one name.

4.1.5 Design Constraints

1: Single-Statement Admission

In order to address the contextual bleed-over issue introduced when injecting contextualized word embeddings into an iterative admission task, our first modification to our original task is to have the model admit or reject only a single statement (the "candidate statement") per training/test instance. When imposed alongside a few other constraints (described below), this change prevents the model from leveraging contextual information gleaned from previous statement extractions at inference time.

2: Cloze-Style Inference Setup

Following constraint **(1)**, and with the goal of providing the model with sufficient information to determine whether or not the single candidate statement should be admitted, we follow practices in classical cloze tasks (Chambers and Jurafsky, 2008, Taylor, 1953) and add to our query set all query-side statements attached to the bottleneck entity. The question presented to the model is, "Given the other relations and events the entity is involved in, does the candidate statement coherently 'fill out'

or 'complete' the entity's representation?" Hence, in **Figure 4.3**, the model's task is to determine whether the "base" entity is located near the Philippines, given that it is known to be located near Nakhon Phanom (an airbase near Thailand), is the location of a set of "forces," and is involved in a set of (admittedly unspecific) "service" and "flown" events.

3: Contextual Overlap Prevention

For positive instances, we ensure that no statement on the query side of the bottleneck overlaps in textual span with any statement on the candidate side of the bottleneck in the original document. This ensures that (i) the candidate statement's admission/rejection will not be informed by any bleed-over context gleaned from the other statements attached to the bottleneck entity and that (ii) no trivially exploitable contextual information can propagate via our graph convolutional network. This condition is trivially satisfied for negative instances, since the query and candidate sides of the bifurcation point are from different source texts.

4: Bottleneck Structure

Lastly, we note that the parsing tool currently in use (Li et al., 2019) does not perform cross-document coreference resolution. As a result, when we create negative cloze-style instances (instances whose statements/EREs are drawn from two distinct source documents), the two component knowledge graphs will merge at one (and only one) entity. In the case of positive instances, however, it is likely that the two sides (candidate and query) of the bottleneck entity will meet at a number of nodes beyond solely the bottleneck entity. Hence, in order to enforce structural similarity between instances of the two classes, we impose the constraint that when creating positive instances, the only node at which the two sides intersect is the bottleneck entity. (Again, this condition is trivially fulfilled by negative instances). This is accomplished by artificially cutting off connections that subvert the single bottleneck path in positive instances.

4.1.6 Instance Construction

Our procedure for creating cloze-style instances largely mirrors the one we employ when creating our salads in **Section 3.3**, with a few exceptions.

Entity Merging For Negative Instances

First, when creating negative instances, our requirement that any entities that are to be merged share an ontology label and at least one name remains the same. For positive instances, no merging is required, since only a single source KG is used. In order to ensure that subgraphs on either side of the bifurcation point are reasonably sized, we also require that

- An entity node used to create a bottleneck in a positive instance is attached to at least two event nodes which themselves have at least one additional neighbor
- Both entity nodes used to create a negative instance bottleneck merge point are attached to at least one event node which itself has at least one additional neighbor

"Reserving" Statements for Candidate-Side Subgraphs in Positive Instances

Note that positive instances, unlike negative ones, require that we impose an artificial bifurcation point. Hence, it is necessary in the generation process to make a decision as to how big the subgraph on either side of the bifurcation can be. Because we eventually limit our graph convolutional network to two statement traversals of reach, we believe it makes sense to have the candidate subgraph extend out two statement traversals from the *first non-bottleneck entity encountered along the candidate subgraph.*

Refer back to **Figure 4.3**; starting from the bottleneck entity, we trace the path beginning at the candidate statement and continue traversing the graph until we encounter our first non-bottleneck entity (named "Philippines"). From there, we "reserve" for the candidate-side subgraph all statements reachable from that entity within two traversals which do not overlap with the candidate statement or query set. To ensure that the two sides of the bifurcation point do not meet at any other place, we artificially cut all statements belonging to an ERE captured by the reserve set which are more than two traversals out.

To create the subgraph on the query side of the bifurcation point, we add statements to the query-side subgraph until a leaf is reached.

Query Set Creation

In order to develop an initial query set of statements for our model to expand, we provide our model with all statements reachable within a specified number of traversals (or "hops") from the bottleneck entity *without going through the candidate statement*. As discussed in **Section 4.1.5**, the result is a setup where all query-side statements attached to the bottleneck entity other than the candidate statement are part of the query set, with some extension of the query set into the rest of the query-side subgraph, depending on the number of traversals specified.

4.2 Model Architecture for Cloze-Style Task (with Contextualized Embeddings)

In this section, we present the architecture of the model we employ when operating on cloze-style intances from Data Set 2. We refer to this model as M_{CLOZE} (for "contextualized language model").

4.2.1 Statement-Dependent Entity Representations

The architecture for our M_{CLOZE} model differs from the architecture of the M_{SALAD} model presented in **Chapter 3** in a couple of keys ways.

First, when moving from Word2Vec (Mikolov et al., 2013a;b;c) embeddings to ELMo embeddings, we believe it makes the most sense to develop representations of entities that are *statement-specific*. In the architecture described in **Section 3.4**, we develop singular embeddings for both EREs and statements. This is possible because a given ERE's representation is fixed; all adjacent statements see the same representation.

However, when we consider a setting in which contextualized embeddings are used, it quickly becomes clear that an entity's representation depends on each of its surrounding statements. Consider **Figure 4.4**; in statement 1, the camp is involved in some attack event, and in statement 2, the camp is involved in some building event.

When using our Word2Vec-based model, the "camp" entity has a single representation which is specific neither to statement 1 nor statement 2. However, when using contextualized ELMo embeddings, the "camp" entity now has one representation *when viewed from the lens of statement 1 and the attack event*, and another *when viewed from the lens of statement 2 and the building event*.

Because an entity's information inherently depends on the context in which the statement being examined says it appears, we move toward a setting in which each statement "sees" a (possibly) unique representation of an entity which depends on the context offered by that statement's textual span. Note that we could also develop a unified representation for an entity by simply taking some aggregation of the entity's statement-dependent representations. We considered this approach, but felt that, for the sake of determining the viability of contextualized embeddings in this setting, using more fine-grained statement-dependent representations was more appropriate.

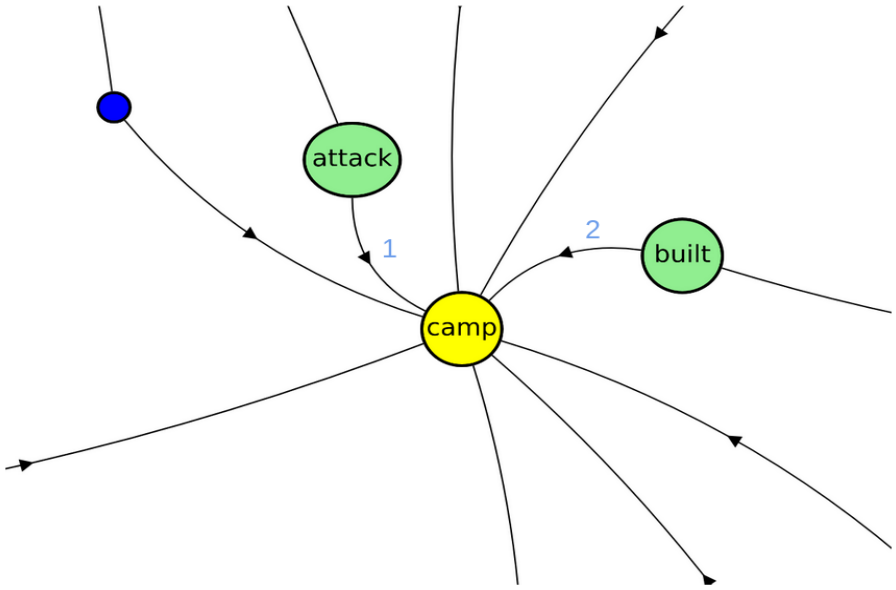

Figure 4.4: Statement-dependent entity example

4.2.2 Model Architecture

Thus, for our ELMo-based model, we no longer develop representations for statements and EREs individually; instead, we only develop explicit representations for statements themselves, *as informed by how they appear in relation to their surround-*

ing event/relations and entities in the source text.

We illustrate our process for developing these representations in **Figure 4.5**. In this case, we are computing a representation for the statement which says that the submarines were the artifact being shipped in this event. To develop a statement-dependent representation of the subject event, we first concatenate the context-invariant embeddings for the ontology labels "movement" and "transport" and the context-specific ELMo embedding for the word "shipped" and feed these through a neural layer and ReLU activation function to produce a "subject embedding." We do the same for the object entity, using the context-invariant embedding for the ontology label "vehicle" and the context-specific ELMo embedding for the word "submarine." We (similarly) feed the context-invariant embedding for the ontology label corresponding to the role that the "submarines" played in the "shipped" event (namely, that of an "artifact") into a neural layer and ReLU activation to produce a "role embedding."

We then concatenate the subject, role, and object embeddings and use a final neural layer and ReLU activation to develop a representation of the associated statement which respects the particular context in which the underlying event and entity were observed. After obtaining a representation via this method for each of the statements in a KG, we then propagate information as before via a GCN, where two statements which share some ERE are processed differently as neighbors based on if the two share a common subject ERE or a common object ERE. A similar attention mechanism is used, with the change that only one context vector is concatenated with each candidate representation, since we no longer have explicit representations of EREs.

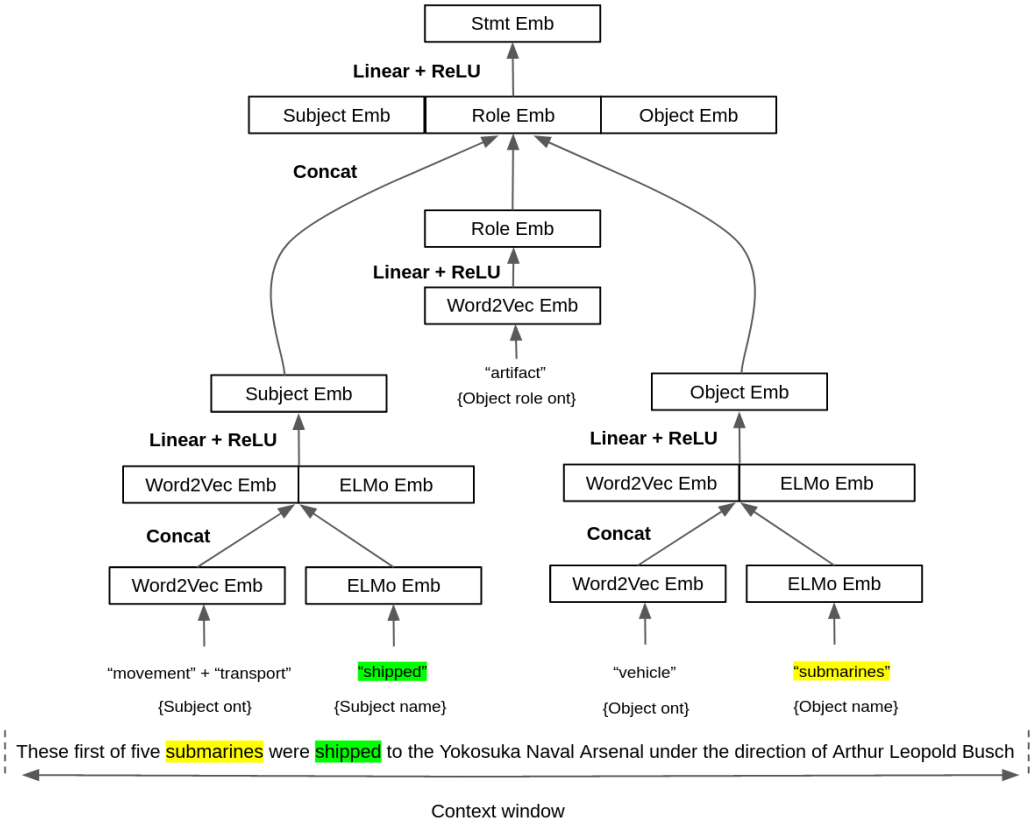

Figure 4.5: Pipeline for entity-dependent statement representations

4.3 Experiments on Data Set 2 (Cloze-Style Data) with M_{CLOZE}

For our initial cloze setting, we train on a total of 160k training instances and test on a total of 20k instances. We validate on a held-out set of 1k instances every 20k training instances. We train for a total of two epochs. Note that the number of training instances in this data set is much higher than in Data Set 1 because we do not impose connectedness or reachability constraints here.

ELMo generates three output representations for each token in a given sentence; the first is the context-invariant output of ELMo's character-level CNN, the second is the first-layer output of ELMo's bi-LSTM, and the third is the second-layer output of the bi-LSTM. The first-layer output of the LSTM tends to capture more local syntactic information, while the second-layer output captures more long-range relationships between tokens (Peters et al., 2018b).

Since this was our first implementation of a model utilizing ELMo embeddings, we experiment with a variety of configurations of the three ELMo outputs. We use a subset of the configurations analyzed in (Reimers and Gurevych, 2019); these include the following variants:

- Unweighted ("fixed") average of all three ELMo output layers
- Only the first-layer, context-invariant ELMo output
- Unweighted ("fixed") average of the first two ELMo output layers

We use 1024-length ELMo embeddings pretrained on the 1 Billion Word Benchmark (Chelba et al., 2013). The ELMo model we used can be found at `https://allennlp.org/elmo`.

Reimers and Gurevych suggest that the third ELMo output layer may be too abstract for some tasks, and can actually lead to reductions in performance in some cases; as a result, we try both an unweighted average of all three ELMo layers and an unweighted average of only the first two layers. As a non-contextualized baseline, we use only the first ELMo output layer, which corresponds to the set of context-invariant token embeddings generated by ELMo's character-level CNN.

We concatenative-style attention for all runs in this experiment. We apply convolution-layer and attention-layer dropout rates of 0.1 and 0.3, respectively, for the cases marked "dropout." A learning rate of 5e-5 is used for all runs.

4.4 Results for Data Set 2 (Cloze-Style Instances)

We first present training curves (in the form of average loss over the training set) for the various ELMo output configurations tested on our initial cloze-style set in **Figure 4.6**. The header "1+2+3 Avg" indicates that an unweighted average of all three ELMo outputs was used, while "Non-context" indicates that only the first, context-invariant ELMo layer was used. Note that we reset the average training loss after every epoch, thus producing the discontinuities/spikes in the plots at each 160k interval.

We similarly present the best recorded loss/precision score for each configuration on the validation set in **Table 4.1**.

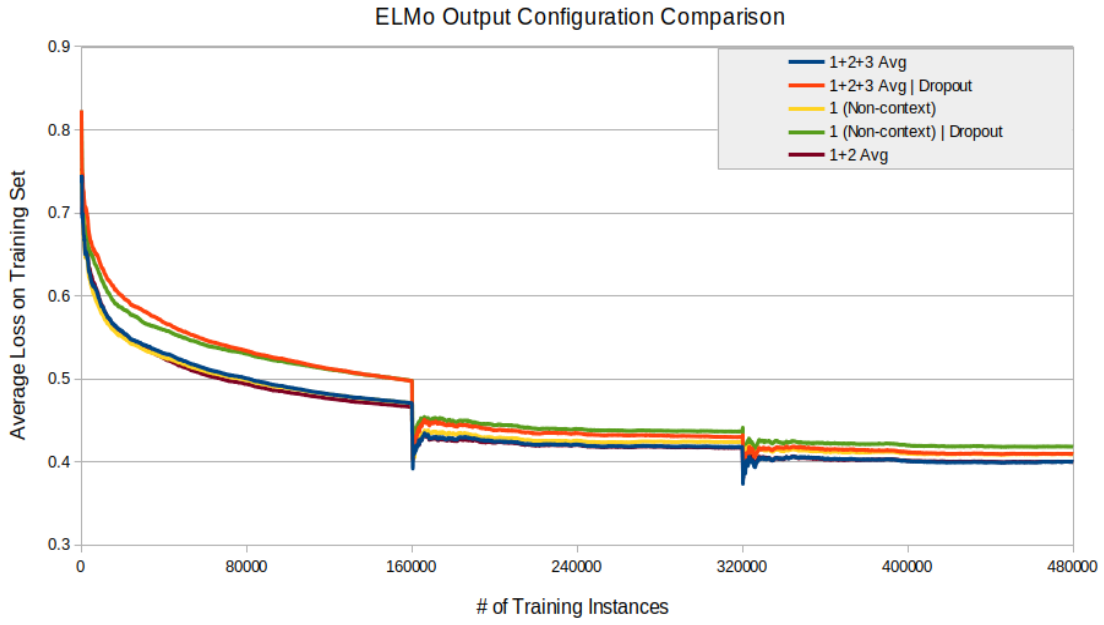

Figure 4.6: ELMo output configuration training curves (average loss) on cloze-style data, where:
1+2+3 Avg = Use an unweighted average of all three ELMo outputs
Dropout = Convolutional-layer dropout of 0.1 and attention-layer dropout of 0.3
1 (Non-context) = Use only the first-layer, context-invariant ELMo output
1+2 Avg = Use an unweighted average of the first two ELMo output layers

	1+2+3 Avg	1+2+3 Avg w/ Dropout	Non-context	Non-context w/ Dropout	1+2 Avg
Loss	.52	.53	.54	.53	.52
Prec	.74	.74	.73	.73	.75

Table 4.1: Average loss/precision on cloze-style validation set for various ELMo output configurations, where:
1+2+3 Avg = Use an unweighted average of all three ELMo outputs
w/ Dropout = Convolutional-layer dropout of 0.1 and attention-layer dropout of 0.3
1 (Non-context) = Use only the first-layer, context-invariant ELMo output
1+2 Avg = Use an unweighted average of the first two ELMo output layers

4.4.1 Remarks on Learnability

First, we note that it appears that our cloze-style set, like our graph salad set, is learnable; the loss decreases fairly smoothly for the configurations tested. The curves seem to level out quite early on, however—epochs 2 and 3 appear to have very marginal impacts (if any) on loss values. This is an observation that carries over into out experiments on graph salads and our experiments on Data Set 2 (the "split-story" cloze-style set); our models for this task in general tend to see disproportionately small gains in performance after the early stages of training.

Additionally, the recorded validation losses and precision scores are all very close in values, reinforcing the notion that the ELMo configurations we tested are all similarly performant.

We take this moment to note that we have had fairly little in the way of systematic human evaluation, barring some interventions to increase node density and move toward structural properties which we think might yield a more learnable task (see **Chapter 3**). Given that we are still grappling with an unwieldy number of design decisions and parameters when creating our data, it would be useful to have some statistically meaningful metric of human performance on our task in the future, if just to confirm that our data is consistent and meaningful to human evaluators. Such metrics might allow us to better probe the reasons behind why our models tend to learn quite slowly, and why performance seems to reach a steady range of values fairly early on.

If we find that human evaluators are largely unable to make sense of mixtures and perform well on the task, for instance, then we might investigate whether our models are simply latching on to trivially exploitable patterns brought about by our data generation process and reaching performance flatlines early on because those methods can only perform so well. Seeing poor human performance might also indicate that our data is inconsistent, and that we need to spend more time ensuring that merged KGs from distinct source documents are actually sufficiently topically different enough to be discernible.

4.4.2 Comparison of Configurations

The models which use a fixed average of either (i) all three ELMo layer outputs or (ii) only the first two ELMo layer outputs appear to perform nearly identically. Both these configurations appear to offer marginal performance increases over their non-contextualized, first-layer-only counterpart. Although some early off-hand experiments with weighted combinations of ELMo outputs, a configuration both initially recommended by (Peters et al., 2018b) and subsequently analyzed in (Reimers and Gurevych, 2019), seemed to suggest little in the way of any changes in performance from unweighted combinations, any future experiments should try these other configurations before any final conclusions are made as to the suitability of the properties of these different output layers for our particular task.

Chapter 5

Data Set 2.5 (Split-Story Cloze-Style Data)

5.1 Data Generation

The cloze-style data generation process just discussed imposes fairly stringent requirements that result in two potential problems. First, heavy pruning (done to eliminate contextual overlap between the candidate and query sides of the bottleneck) leads to the loss of ERE's and statements present in the original graph. In particular, recall that we prune away any statements which link an ERE on the candidate side to an ERE on the query side. This means that, should candidate-side ERE's participate in events or relations which are also connected to query-side ERE's, those connections will not be present in the data set. As a result, many points of commonality between the candidate and query subgraphs are removed.

Second, our requirement that the candidate side be attached to the bottleneck entity by one, and only one, statement (namely, the candidate statement) means that candidate-side subgraphs are likely to be smaller in size, and thus less information-rich, than the query-side subgraphs, which receive all the other bottleneck-adjacent statements. Refer (again) back to **Figure 4.3**; only the statement linking the base to the Philippines is assigned to the candidate-side subgraph, leaving all other statements adjacent to the bottleneck for the query side.

We present a modified version of our cloze-style data in which we aim to address these issues. We detail the ways in which this modified data set differs from our initial cloze-style set in the sections below; we refer to this variant of cloze-style data as "split-story" data, so-named because the resulting instances tend to have more equal-sized candidate-side and query-side subgraphs.

Node Duplication

First, instead of removing connections that circumvent the bottleneck entity, we duplicate nodes on either end of the bottleneck and assign any contextually problematic statements to either the candidate or query side whenever possible. While not all statements can be preserved in this manner due to some edge cases which we

refrain from detailing here, this change allows us to keep more of the original graph structure intact than under the method described in **Section 4.1.6**.

With this duplication trick, we preserve on average 3.5 statements per cloze-style instance in our 70k training set. Note that, while this number is fairly small, these statements can critically establish common entities/events between the two sides of the bottleneck. Additionally, we recorded a case in which 39% of all statements in an instance were preserved using this trick. Although 46,990 instances did not preserve additional statements as a results of this trick, cases like the above indicate the impact that duplication can have on keeping a graph intact.

Bottleneck Statement Assignment

Second, we distribute as evenly as possible the set of statements attached to the bottleneck entity between the candidate and query sides. Recall that, under the method outlined in **Section 4.1.6**, we previously assigned only one bottleneck-adjacent statement to the candidate subgraph. As with the pruning of connections, this detail also led to generally smaller candidate-side subgraphs, while ensuring that query-side subgraphs were generally larger. More evenly distributing bottleneck-adjacent statements among the candidate and query subgraphs, along with the duplication trick, leads to higher parity between candidate and query subgraph sizes.

We accomplish this by alternately assigning bottleneck-adjacent statements to the candidate and query sides; should we assign a given bottleneck-adjacent statement to the candidate-side subgraph which overlaps in textual span with some of the still-unassigned bottleneck-adjacent statements, those statements are all added to the candidate side at the same time. (The same goes for bottleneck-adjacent statements which are assigned to the query side.) Hence, our constraint that statements on the candidate side cannot overlap in textual span with statements on the query side (and vice-versa) means that a perfect "50/50" assignment scheme of bottleneck-adjacent statements cannot always be achieved.

An example of an instance whose bottleneck-adjacent statements were assigned using this scheme is shown in **Figure 5.1**.

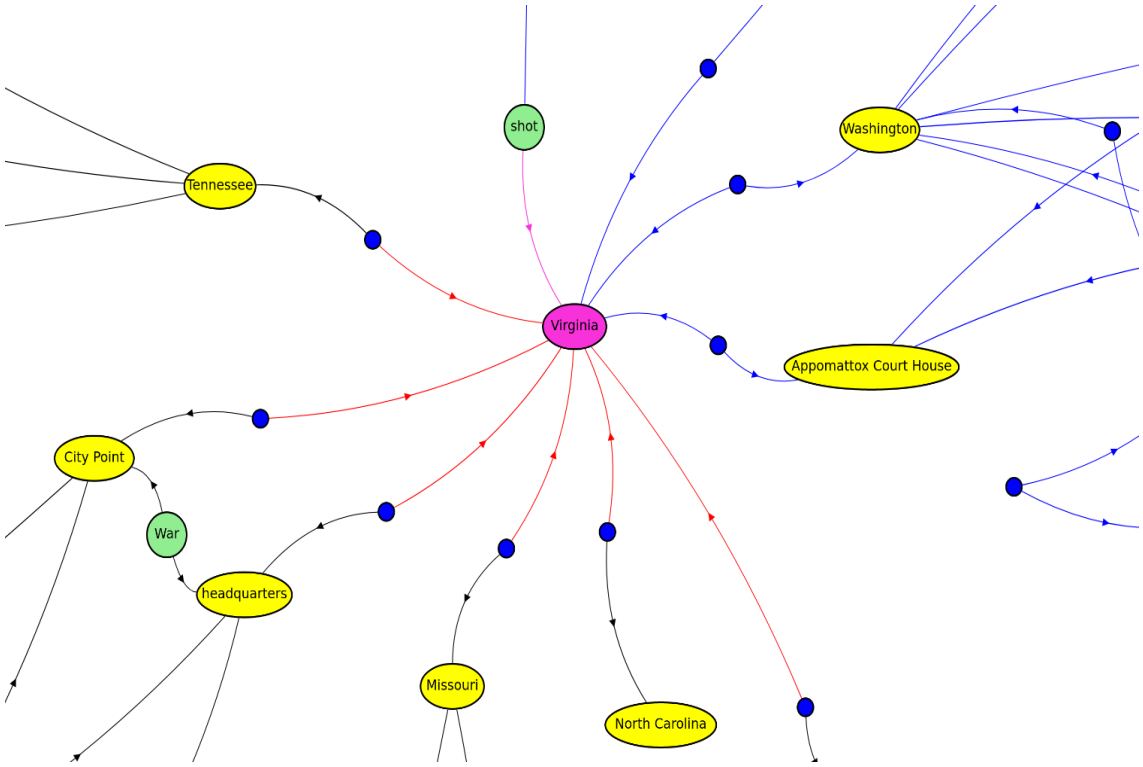

Figure 5.1: "Split-story" cloze-style example

Bottleneck Connectedness Requirements

We enforce a minimum total one-step and two-step connectedness of 4 and 8, respectively, for the bottleneck entity. Note that a similar connectedness constraint was used for our graph salads. We add this constraint to the "split-story" variant set in an attempt to generate a data set whose decision point is located in denser regions of graphs (and, therefore, regions which are likely to be of more salience to the knowledge graphs and narratives).

Candidate Statement Type

Lastly, while neither the candidate nor the query subgraph were required to contain an event statement adjacent to the bottleneck entity in Data Set 2, we require in our "split-story" set that the candidate statement be an event statement. After an evaluation of Data Set 2, we concluded that relation statements were too difficult to predict. We initially tried to require that both the candidate and query sides have

at least one bottleneck-adjacent event, but found that this reduced our data set size too drastically.

5.2 Modeling

We apply our M_{CLOZE} model from **Chapter 4** to our split-story data set, making one architectural change in the process. As previously discussed, our "split-story" variant of the cloze-style data generally yields more sizeable candidate subgraphs. As a means of exploring the usefulness of the additional narrative content which longer-range contextualization may afford, we implement one architectural addition to the M_{CLOZE} model during our experiments on the "split-story" cloze-style data.

First, we compute a weighted sum of all GCN statement embeddings on the query side of the bottleneck. We employ a distance-informed heuristic when assigning weights; statements one traversal away from the bottleneck (i.e., adjacent to the bottleneck) are assigned a weight of 0.35, statements two traversals away 0.25, and so on. The same is done for the candidate-side subgraph. (Note: the exact weighting schemas for subgraphs of varying sizes are specified in **Section A.3.2** in the appendices.)

We then concatenate each query-set statement's GCN representation with the weighted sum of query-side embeddings, and (similarly) concatenate the candidate statement's GCN representation with the weighted sum of candidate-side embeddings. We feed these into a tanh layer before the attention process.

5.3 Experiments on Data Set 2.5 ("Split-Story" Cloze-Style Data)

After updating our data set to support node duplication, multiple candidate-side event bottleneck statements, and more sizeable candidate subgraphs, we train an M_{CLOZE} model which aims to leverage the additional context afforded by this data. As mentioned in **Section 5.2**, for a subset of our runs, we fold in a weighted aggregation of either side of the bottleneck as additional context during learning.

We train on a total of 70k training instances for four epochs and test on a total of 9k instances, validating our model on a held-out set of 5k instances every 17,500

training instances. We use an unweighted average of all three ELMo outputs for our contextualized embeddings, as we found that there was little difference in performance between this configuration and an unweighted average of only the first two ELMo outputs (see **Section 4.4**). Note that our data set size is considerably less than it is for Data Set 2 because the additional connectedness constraints, along with the requirement that the candidate statement be an event statement, limit the number of graphs we can produce using this method.

We try the following:

- Use query-side and candidate-side subgraph context vectors
- Vary the maximum number of hops in the query set (relative to the bottleneck) from 1 to 3
- Compare the efficacy of using (i) a fixed average of all three ELMo output token representations and (ii) only the first, context-invariant ELMo output layer (our baseline)
- Experiment with dropout rates

Varying the size of the query statements by changing the value of x is intended to provide insight as to whether having a larger narrative-so-far subgraph has an effect on learning. As in the experiment for data set 2 above, a comparison of (i) a fixed average of all three ELMo output layers and (ii) only the first ELMo output layer allows us to determine whether contextualized word embeddings offer any marked performance increases over their non-contextualized counterparts in our task. Lastly, we try ignoring all name information (e.g., "Sandinista," "Nicaragua") and using only ontology labels (e.g, "Organization," "GeopoliticalEntity") in our representations to determine the impact of name-related information on learning.

We apply dropout only to the model variants presented in the "dropout" experiment.

5.4 Results for Data Set 2.5 (Split-Story Cloze-Style Instances)

We conduct a total of 24 training runs on Data Set 2.5. So as to offer clear and direct comparisons, we divide these results into sections corresponding to each ablation variable, as follows:

- Use of query/candidate-side context vectors
- Fixed average of all 3 ELMo outputs vs. just the first-layer output (contextualized vs. non-contextualized)
- Use of ERE name information
- Dropout

For each section, we select a single comparison plot at hand to illustrate results. A full list of plots for our experiments on Data Set 2.5 can be found in **Section B.1** in the appendices; for example, while a plot comparing models which do/do not leverage additional subgraph context vectors is presented only in the context of models which use both name and ontoloy information, a similar comparison plot displaying training curves for models which use only ontology information is shown in **Figure B.2** in **Section B.1** in the appendices.

Note that, unless otherwise specified by the legend/experiment configurations, the models to follow in the below sections use concatenative-style attention, use a fixed/unweighted average of all three ELMo layers, leverage both name and ontology ERE information, and do not make use of subgraph context vectors.

5.4.1 Query/Candidate-side Context Vectors

As discussed in **Section 5.3**, training occurs over a total of 70k instances for 4 epochs. As in previous experiments, the average training loss (plotted on the y-axis) is reset at the end of each epoch, thus producing sudden oscillations in the plots at each 70k interval.

In **Figure 5.2**, we show results for models which either use/do not use additional subgraph context vectors when the set of query statements extends out a maximum of 1, 2, and 3 hops from the bottleneck entity, where a "hop" is a single statement traversal. In other words, in the "1 Hop" case, all query-side statements which are adjacent to the bottleneck entity are considered to be part of the query set (and, thus, participate as attendees during the attention process). Similarly, in the "2 Hops | Subgraph," all query-side statements reachable within two hops of the bottleneck entity constitute the query set, and the model additionally leverages subgraph context

vectors. **Table 5.1** similarly displays final precision scores on the test set for these configurations.

In all cases tested, the model which leverages query- and candidate-side context vectors outperforms the model which does not. The difference in performance appears to become more pronounced as the number of hops in the query set increases.

We summarize this effect in **Table 5.2**, in which we display the differences in final precision averages over the test set between models which use the subgraph context vectors and models which do not (e.g., the model using concatenative attention, a fixed average of the three ELMo outputs, and only ontology information saw an increase in its average precision score on the training set of +0.023 at the conclusion of training when also using subgraph context vectors).

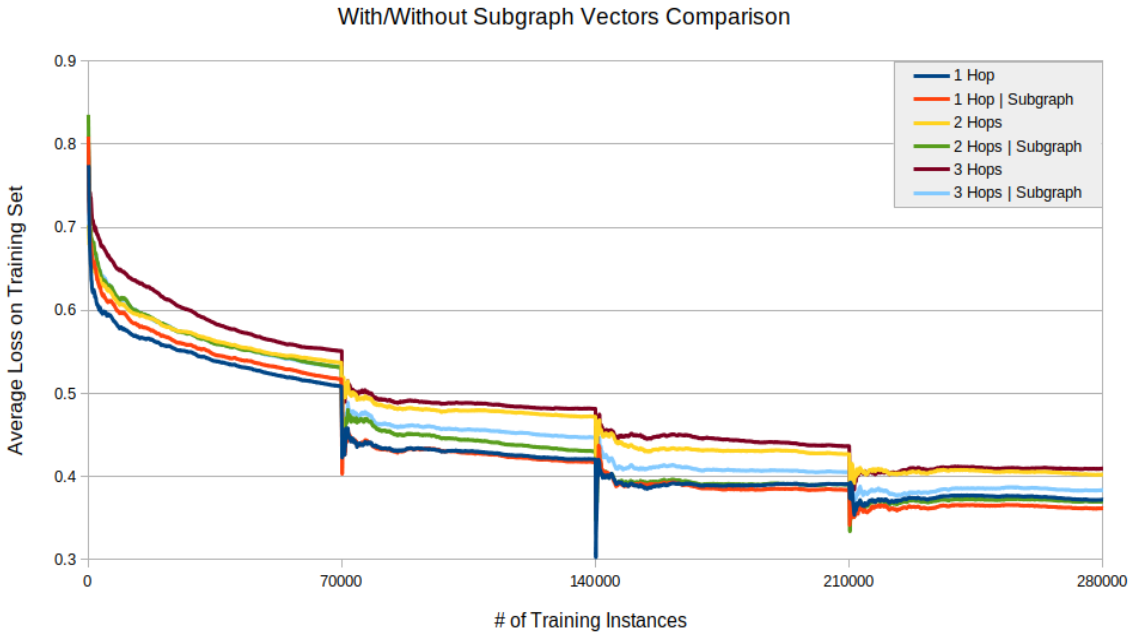

Figure 5.2: Training curves (average loss) on split-story cloze-style data for models with/without subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Subgraph = Use subgraph context vectors

	1 Hop	2 Hops	3 Hops
Non-subgraph	.77	.76	.76
Subgraph	.79	.78	.78

Table 5.1: Average precision on split-story cloze-style test set for models with/without subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-subgraph = Does not use subgraph context vectors
Subgraph = Uses subgraph context vectors

	1+2+3 Avg		First Layer
	Names + Ont	Ont Only	Names + Ont
1 Hop	.004	.012	.008
2 Hops	.018	.023	.012
3 Hops	.014	.025	—

Table 5.2: Δ average precision (+) on split-story cloze-style test set when using subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = Uses an unweighted average of all three ELMo outputs
First Layer = Uses only the first-layer, context-invariant ELMo output
Names + Ont = Uses both name information and ontology labels
Ont Only = Uses only ontology labels

Given the clear (if marginal) performance gain afforded by using subgraph context vectors, further investigation into ways of allowing for more long-range information-passing is worthwhile. We currently remain limited by the short-range nature of information passing in GCNs (Marcheggiani and Titov, 2017, Chen et al., 2019), thus restricting context in our KGs to localized regions of information around the bottleneck entity which can be insufficient for inference. We discuss potential ways of exploring this in the **Future Work** chapter.

5.4.2 Contextualized vs. Non-contextualized

In **Figure 5.3**, we compare the training curves of models which use either (i) a fixed average of all three ELMo output token embeddings (the "contextualized" case) or (ii) only the context-invariant first-layer ELMo output (the "non-contextualized" case). **Table 5.3** similarly shows final precision scores on the test set for these variants.

Clearly, contextualized embeddings offer a performance increase (if slight) over non-contextualized embeddings for this data set. One might expect more substantial discrepancies in performance, given the importance of contextualization to our narrative-driven task; it remains unclear why we do not observe a more substantial difference. As discussed in **Section 4.4.2**, further experiments with various configurations of the ELMo layers might help to determine if the ELMo output layers which *are* contextualized might be leveraged in a better way.

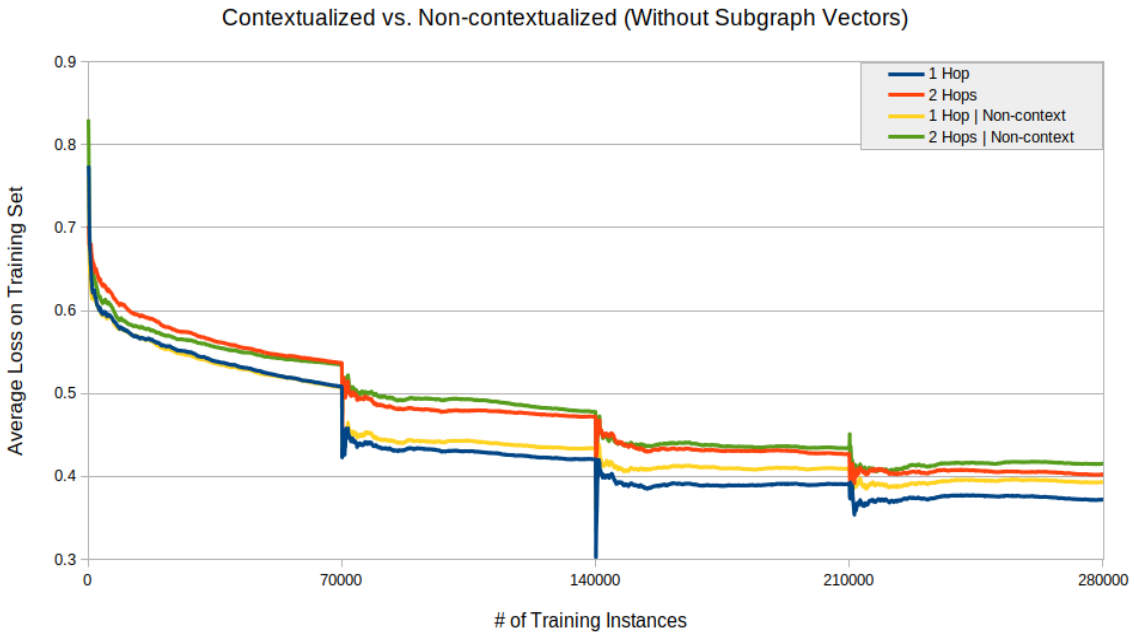

Figure 5.3: Training curves (average loss) on split-story cloze-style data for models using either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-context = only the first-layer, context-invariant output of ELMo

	1 Hop	2 Hops
1+2+3 Avg	.77	.76
First Layer	.77	.75

Table 5.3: Average precision on split-story cloze-style test set for models using either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = unweighted average of all three ELMo outputs
First Layer = only the first-layer, context-invariant output of ELMo

5.4.3 Using Name and Ontology Information vs. Ontology Information Only

In **Figure 5.4**, we present plots which compare the training curves of models which leverage both name (e.g., "Sandinista") and ontology (e.g., "GeopoliticalEntity") information and models which leverage only ontology information. **Table 5.4** displays the final precision scores on the test set for these configurations.

Given the marked performance decreases when using only ontology labels, it is clear that name labels offer an important source of enriching context and serve as a critical supplement to ontology labels.

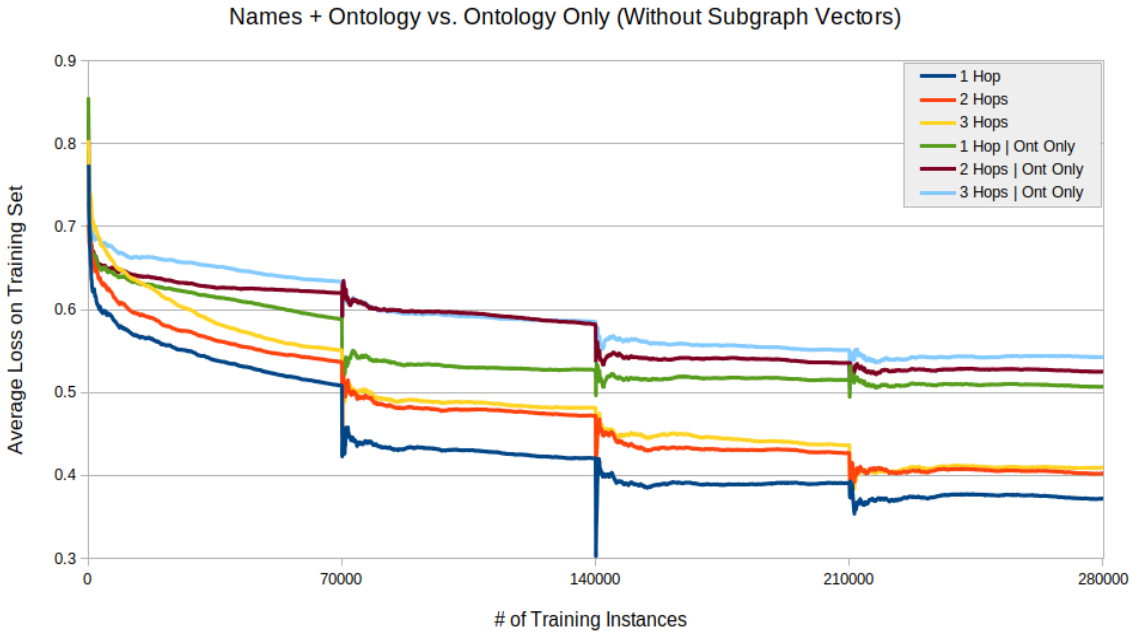

Figure 5.4: Training curves (average loss) on split-story cloze-style data for models using either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Ont only = ontology labels alone

	1 Hop	2 Hops	3 Hops
Name + Ont	.77	.76	.76
Ont Only	.71	.70	.68

Table 5.4: Average precision on split-story cloze-style test set for models using either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Name + Ont = name information and ontology labels
Ont only = ontology labels alone

5.4.4 Dropout

Lastly, we compare configurations of dropout rates. All models pictured in **Figure 5.5** use subgraph context vectors and a query set consisting of all query-side statements adjacent to the bottleneck entity (our best-performing configuration). The legend entries are in the form "Conv: x | Attn: y," where x is the probability of applying dropout to GCN layers, and y is the probability of applying dropout to attention layers.

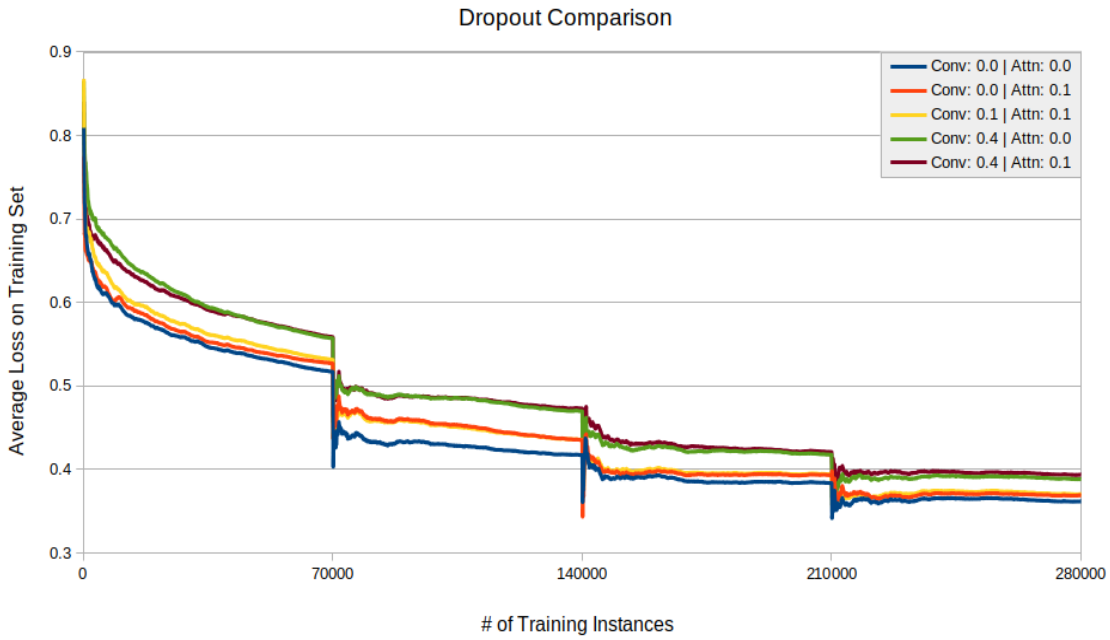

Figure 5.5: Training curves (average loss) on split-story cloze-style instances for models with varying dropout configurations, where:
Conv: x / Attn: y means x is the probability of applying dropout to GCN layers, and y is the probability of applying dropout to attention layers

We similarly plot the final average loss and precision values on the test set of these variants in **Table 5.5**. We initially refrained from applying dropout to our cloze-style models because early observations seemed to suggest that dropout was overly detrimental to our model's ability to learn; along these lines, we were also careful with the amount of dropout we applied during this experiment (e.g., trying both a convolution-layer dropout rate of 0.0 and 0.1).

Given that the five dropout variants appear to perform very comparably on the test set, our initial suspicion that dropout was problematic for these models seems to be refuted, and, given that regularization helps to improve generalizability, future experiments should include some dropout and further experimentation with dropout rates or other forms of regularization.

	Avg Loss	Avg Precision
Conv: 0.0 | Attn: 0.0	.46	.79
Conv: 0.0 | Attn: 0.1	.45	.79
Conv: 0.1 | Attn: 0.1	.46	.78
Conv: 0.4 | Attn: 0.0	.47	.78
Conv: 0.4 | Attn: 0.1	.47	.77

Table 5.5: Average precision on split-story cloze-style test set for models with varying dropout configurations, where:
Conv: x / Attn: y means x is the probability of applying dropout to GCN layers, and y is the probability of applying dropout to attention layers

5.4.5 Test Set Results

We present the average loss and precision values on the test set of the best-performing validation checkpoints for almost all model variants in **Figures 5.6** and **5.7**. (Only the dropout experiments are excluded, given that they do not fit neatly into tables of this style.)

These results summarize our previous observations; concatentative attention is more suited to our task than bilinear attention, subgraph context vectors help across the board, taking a fixed average of the three ELMo outputs offers marginal performance benefits over using only the non-contextualized first-layer output, and context-rich name information is a critical supplement to ontology labels. Additionally, we note here that increasing the size of the query set (in particular, going from a maximum of 1 to 2 hops) seems to slightly decrease performance.

Most of the values presented in **Figures 5.6** and **5.7** (and, for that matter, in the training curves presented earlier) are generally quite close. Hence, we note that the observations just presented are only true to a fairly marginal degree; the difference between our best- and worst-performing models is only around 10 points of precision,

with differences in some of the less impactful comparisons (like using an unweighted average of all three ELMo outputs vs. using just the non-contextualized first-layer output) being on the order of 1 to 0.5 points of precision.

			Max Query Distance		
			1 Hop	2 Hops	3 Hops
1+2+3 Avg	Names + Ont	Subgraph	.45	.45	.45
		Non-subgraph	.48	.50	.50
	Ont Only	Subgraph	.52	.51	.53
		Non-subgraph	.54	.55	.57
First Layer	Names + Ont	Subgraph	.45	.47	
		Non-subgraph	.48	.52	

Table 5.6: Average loss on split-story cloze-style test set for various models, where:
Max Query Distance: x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = an unweighted average of all three ELMo outputs ("contextualized")
Names + Ont = Both name information and ontology labels are used
Ont Only = Only ontology labels are used
Subgraph = Subgraph context vectors are used
Non-subgraph = Subgraph context vectors are not used

			Max Query Distance		
			1 Hop	2 Hops	3 Hops
1+2+3 Avg	Names + Ont	Subgraph	.79	.78	.78
		Non-subgraph	.77	.76	.76
	Ont Only	Subgraph	.73	.73	.71
		Non-subgraph	.71	.70	.68
First Layer	Names + Ont	Subgraph	.78	.77	
		Non-subgraph	.77	.75	

Table 5.7: Average precision on split-story cloze-style test set for various models, where:
Max Query Distance: x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = an unweighted average of all three ELMo outputs ("contextualized")
Names + Ont = Both name information and ontology labels are used
Ont Only = Only ontology labels are used
Subgraph = Subgraph context vectors are used
Non-subgraph = Subgraph context vectors are not used

5.4.6 Bottleneck Characteristic Comparison

Finally, we isolate subsets of the test data with particular characteristics and report related performance metrics.

Subset Isolation

First, we isolate cloze-style instances which have a bottleneck entity name consisting of all lowercase letters from those which have bottleneck entity names containing at least one uppercase letter. We compare these data subsets to determine whether bottleneck entities with more specific names (e.g., "J. Edgar Hoover") are more or less difficult for inference than bottleneck entities with more generic names (e.g., "road").

We similarly isolate cloze-style instances into subsets consisting of the 2,250 (25% of our test set) most highly connected bottleneck entities and the 6,750 (75%) remaining instances. Here, we seek to determine the effect of high-density decision points on learning.

In **Tables 5.8** and **5.9**, we present subset-specific performance disparities. For

rows marked "lowercase entity name," the values indicate the difference in loss/precision when evaluating on only instances with lowercase bottleneck entities vs. evaluating on instances with a capital letter in bottleneck names. For rows marked "less connected entity," the values indicate the difference in loss/precision when evaluating on instances containing only the bottom 6,750 least connected entities vs. evaluating on instances containing the remaining 2,250 most highly connected entities. For both figures, the third row indicates the change when evaluating on entities which meet both of the specified criteria vs. on entities which meet neither.

As an example, model variants performed better on instances with entirely lowercase bottleneck names by an average of 2 precision points than they did on instances with bottleneck names containing a capital letter. Similarly, model variants performed better on the 6,750 instances with less connected bottlenecks by an average of 4 precision points than they did on the 2,250 instances with the most highly connected bottlenecks. Lastly, model variants performed better on instances which had a less connected bottleneck with an entirely lowercase name by an average of 4 precision points than they did on instances which had a highly connected bottleneck with a capital letter in its name.

Given the importance of contextualized name information to an entity's specificity, we similarly partition runs into those which use both name and ontology labels ("contextualized") and those which use only ontology labels ("non-contextualized") for easy analysis. The "all runs" column takes into account both of those subsets.

	All Runs	Name + Ont Runs	Ont Only
Lowercase Entity Name	-.04	-.09	.04
Less Connected Entity	-.09	-.08	-.10
Both Above Conditions	-.09	-.09	-.09

Table 5.8: Δ average loss on split-story cloze-style test set for subsets of bottleneck entities, where:
Lowercase Entity Name = change when operating on subset of instances whose bottleneck entities do not contain a capital letter
Less Connected Entity = change when operating on subset of instances whose bottleneck entities are among the 7,750 less connected bottlenecks in the test set
Both Above Conditions = change when operating on subset of instances whose bottleneck entities (i) contain no capital letters and (ii) are among the 7,750 less connected bottlenecks in the test set
Name + Ont Runs = results for only runs using both name information and ontology labels
Ont Only = results for only runs which use ontology labels alone

	All Runs	Name + Ont Runs	Ont Only
Lowercase Entity Name	.02	.06	-.05
Less Connected Entity	.04	.04	.05
Both Above Conditions	.04	.04	.04

Table 5.9: Δ average precision on split-story cloze-style test set for subsets of bottleneck entities, where:
Lowercase Entity Name = change when operating on subset of instances whose bottleneck entities do not contain a capital letter
Less Connected Entity = change when operating on subset of instances whose bottleneck entities are among the 7,750 less connected bottlenecks in the test set
Both Above Conditions = change when operating on subset of instances whose bottleneck entities (i) contain no capital letters and (ii) are among the 7,750 less connected bottlenecks in the test set
Name + Ont Runs = results for only runs using both name information and ontology labels
Ont Only = results for only runs which use ontology labels alone

Observations

For runs using both name and ontology information, we observe a sizeable increase of 6 points of precision when considering only entirely lowercase bottlenecks. Although this "lowercase" condition is only a rough heuristic suggestion of entity specificity, this seems to suggest that inference is easier for more generic bottleneck entities. Further investigation is required and warranted for any well-founded conclusions, but one possible theory here is that, because more specific entities tend to be tied to more specific contexts (e.g., a mention of General Pickett generally accompanies texts describing the Battle of Gettysburg), the documents our data generation procedure selects for mixing are more likely to be topically similar when entity names are more specific. (On the other end of the spectrum, the word "road" appears in an innumerable number of documents and scenarios.)

For runs which leverage only ontology labels, we see an average decrease of 5 precision points when considering only less generic bottleneck names. Although the reason for the decrease is unclear, it makes sense that ontology-only runs do not see an increase in precision here; by the proposed reasoning in the previous paragraph, they would not be as attentive to topical similarity among different merged documents when entities are more specific, and would (thus) be less likely to see an increase in performance when considering only more generic entities.

Lastly, we note that all runs (leveraging either name and ontology information or ontology labels alone) see performance increases when considering only the 6,750 least connected bottlenecks. This result seems reasonable, as highly-dense bottlenecks are more likely to complicate information flow in GCNs than less dense bottlenecks (i.e., more dense bottlenecks have more competing information pathways feeding into them from the candidate and query sides than less dense bottlenecks, which may complicate inference).

Chapter 6

Conclusion and Future Work

6.1 Recap of Work

In this work, we sought to determine if neural models are capable of disentangling mixtures of narratives drawn from texts which present mutually conflicting stories. Because there is currently no gold-label data for this task, we created and presented two main "flavors" of synthetic data using Wikipedia articles under the categories *war/conflict*. We showed that all of our data sets are learnable by effectively training a GCN-based, attention-informed neural architecture in a coherence-based inference task.

Our first data set consists of graph salads, wherein distinct source knowledge graphs are artificially merged at multiple points and our model is asked to iteratively add statements to a seed narrative subgraph. Because we theorized that simply injecting contextualized word embeddings to enrich representations for our graph salads would be unviable due to an issue we refer to as "contextual bleed-over," we developed an additional type of "cloze-style" data which has distinct source documents meet at only a single point (the "bottleneck entity"), and which asks our model to admit or reject only a single statement per training instance. We went on to better preserve original graph structures when creating cloze-style instances in our "split-story" variant of cloze-style data, and showed that longer-range contextualization can benefit inference in our task by using a simple weighted aggregation of both sides of the bottleneck entity in each instance.

While we attempted to frame our task in an RL-based setting, we found that unclear code-level implementation details required for our algorithm of choice to work effectively prevented our RL extension from yielding positive results.

6.2 Future Work

6.2.1 Data Generation

While our data is learnable, there are still numerous avenues to pursue in order to better simulate a setting in which multiple narratives share a set of common entities/events, but are mutually contradictory. First, as discussed in **Section 3.3.7**, our-name matching heuristic is currently a simple solution to merging entities and events, and can often produce either trivial or impossible examples, likely depending on the specificity of the names of the nodes merged.

Our results for our data subset analysis in **Section 5.4.6** appear to support this theory; our models generally see an improvement in precision scores when operating on KG mixtures which (i) meet at bottleneck entities with names containing all lowercase letters or (ii) are in less-dense regions of the graph. We believe that these results motivate additional investigation into our method of merging entities; named entity recognition systems might be leveraged to better separate out specific entities/events (e.g., "Julius Caesar") from their more generic counterparts (e.g., "boat") for a more accurate analysis of how specificity can affect the challenge of our task.

Similarly, despite our efforts to increase statement variety around merge points, the fact remains that merge points are, on the whole, still few and far between; future efforts might explore the idea presented in **Section 3.3.8**, which entails introducing more "noise" into our mixtures by applying a less-constrained merging process to areas of mixtures which lack narrative diversity.

Lastly, while we do operate on KGs drawn from multimodal sources during AIDA evaluations, our synthetic data is still composed entirely of graphs drawn from text documents. Future experiments might seek to leverage document timestamp and geostamp data as a means of taking a small but approachable step toward the longer-term goal of folding in non-text-based sources of information into our synthetic data.

6.2.2 Modeling

We reiterate that the GCN component of our network limits the distance which information can propagate throughout our KGs during convolution. Additional work on this task should investigate methods which allow for less constrained contextual-

ization; approaches like those in (Veličković et al., 2017, Chen et al., 2019) suggest that self-attention can allow richer means of contextualization, particularly when nodes are allowed to attend to all other nodes in a KG (Chen et al., 2019). By using path-based information between nodes to inform attention (as Chen et al. do), such methods might allow us to represent longer-range dependencies without the use of RL, which has proven problematic for us.

Appendix A

Hyperparameter Settings

A.1 M_{SALAD} (Model for Graph Salads)

- Learning rate: 1e-5
- Hidden/attention vector size: 300
- Number of GCN layers: 2
- Batch size: 25
- Number of training epochs: 2
- Teacher forcing probability (initial): 0.25
- Teacher forcing decay rate: 0.9659 every 3k training salads
- Dropout for GCN layers: 0.5
- Dropout for attention layer: 0.3
- Optimizer: Adam (Kingma and Ba, 2015)
- Weight decay: 0.001

A.2 $M_{SALAD\text{-}RL}$ (Model for Graph Salads with RL Extension)

A.2.1 Actor Pretraining

- Learning rate: 1e-5
- Hidden/attention vector size: 300
- Batch size: 5
- Number of training epochs: 2
- Teacher forcing probability (initial): 0.95

- Teacher forcing decay rate: 0.9768 every 1k training salads
- Dropout for GCN layers: 0.5
- Dropout for attention layer: 0.2
- Optimizer: Adam (Kingma and Ba, 2015)
- Weight decay: 0.5

A.2.2 Critic Training/PPO-Specific Parameters

- Learning rate: 1e-5 | 1e-4
- Hidden/attention vector size: 300
- Batch size: 5
- Number of training episodes: 120k
- Dropout for GCN layers: 0.5
- Dropout for attention layer: 0.2
- Trajectory length: 25 statement admissions (max)
- Epsilon clip value: 0.2
- Optimizer: Adam (Kingma and Ba, 2015)
- Weight decay: 0.5
- Discount factor (γ): 0.99
- Trace decay rate (λ): 0.9

A.3 M_{CLOZE} (ELMo-Based Model for Cloze-Style Instances)

A.3.1 Model for Data Set 2

(Original Cloze-Style Instances)

- Learning rate: 5e-5
- Hidden/attention vector size: 600
- Number of GCN layers: 2
- Batch size: 25
- Number of training epochs: 2
- Dropout for GCN layers: 0
- Dropout for attention layer: 0
- Optimizer: Adam (Kingma and Ba, 2015)

A.3.2 Model for Data Set 2.5

("Split-Story" Cloze-Style Instances)

- Learning rate: 5e-5
- Hidden/attention vector size: 600
- Number of GCN layers: 2
- Batch size: 25
- Number of training epochs: 4
- Dropout for GCN layers (except for dropout experiments): 0
- Dropout for attention layer (except for dropout experiments): 0
- Optimizer: Adam (Kingma and Ba, 2015)

- Weighting schemas for candidate/query subgraph context vectors (note: weights are presented in the form *x: weight*, where x is the minimum number of traversals from the bottleneck entity to a given statement; i.e., read "1: .35" as "statements one traversal from the bottleneck entity are assigned a weight of .35")
  - For subgraphs with paths at most 6 traversals from the bottleneck entity: {1: .35, 2: .25, 3: .20, 4: .10, 5: .06, 6: .04}
  - For subgraphs with paths at most 5 traversals from the bottleneck entity: {1: .358, 2: .258, 3: .208, 4: .108, 5: .068}
  - For subgraphs with paths at most 4 traversals from the bottleneck entity: {1: .375, 2: .275, 3: .225, 4: .125}
  - For subgraphs with paths at most 3 traversals from the bottleneck entity: {1: $.41\overline{6}$, 2: $.31\overline{6}$, 3: $.2\overline{6}$}
  - For subgraphs with paths at most 2 traversals from the bottleneck entity: {1: .55, 2: .45}
  - For subgraphs with paths at most 1 traversals from the bottleneck entity: {1: 1.0}

Appendix B

All Data Set 2.5 (Split-story Cloze-Style) Figures and Tables

B.1 Subgraph Context Vectors

As before, we note that in all cases, subgraph context vectors offer a modest increase in performance.

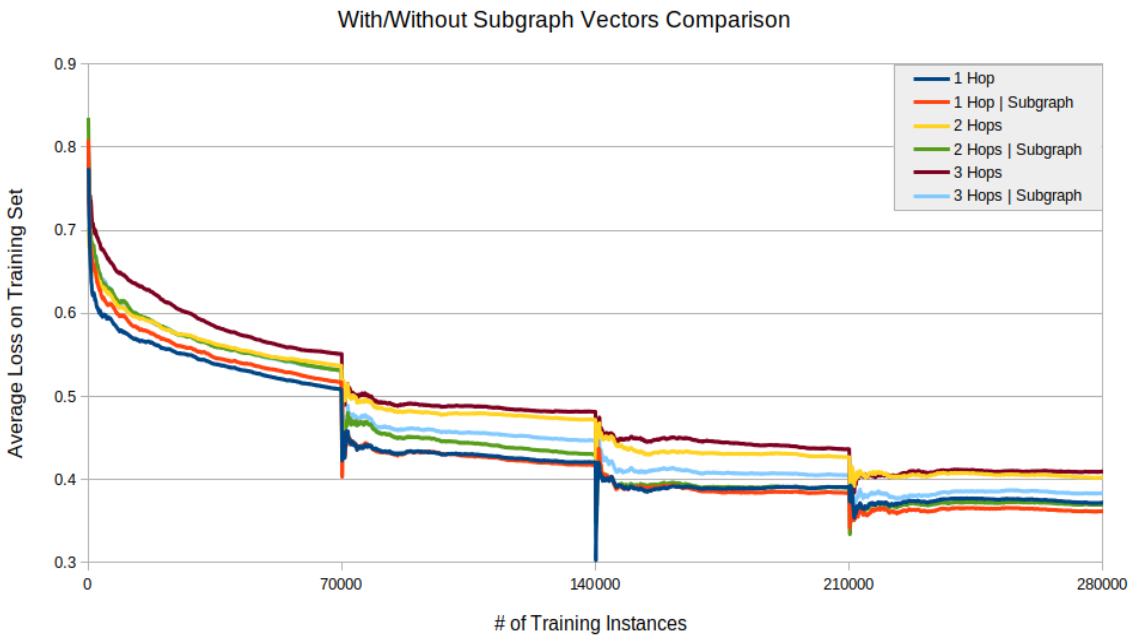

Figure B.1: Training curves (average loss) on split-story cloze-style data for models with/without subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Subgraph = Use subgraph context vectors

	1 Hop	2 Hops	3 Hops
Non-subgraph	.77	.76	.76
Subgraph	.79	.78	.78

Table B.1: Average precision on split-story cloze-style test set for models with/without subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-subgraph = Does not use subgraph context vectors
Subgraph = Uses subgraph context vectors

Figure B.1 and **Table B.1** compare training loss curves and test precision scores, respectively, for models which either do/do not use additional subgraph context vectors.

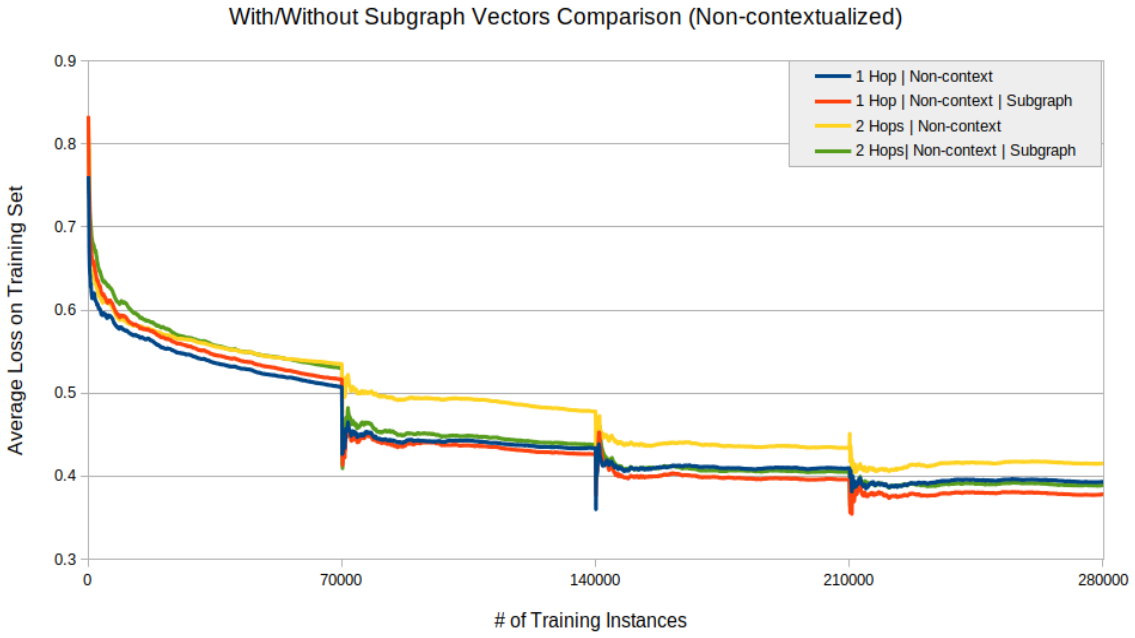

Figure B.2: Training curves (average loss) on split-story cloze-style data for models with/without subgraph context vectors (using only the first-layer, context-invariant ELMo output), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Subgraph = Use subgraph context vectors
Non-context = only the first-layer, context-invariant ELMo output

	1 Hop	2 Hops
Non-subgraph	.77	.75
Subgraph	.78	.77

Table B.2: Average precision on split-story cloze-style data for models with/without subgraph context vectors (using only the first-layer, context-invariant ELMo output), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-subgraph = do not use subgraph context vectors
Subgraph = use subgraph context vectors

Figure B.2 and **Table B.2** compare training loss curves and test precision scores, respectively, for models which either do/do not use additional subgraph context vectors *with concatenative attention and only the non-contextualized first-layer ELMo output.*

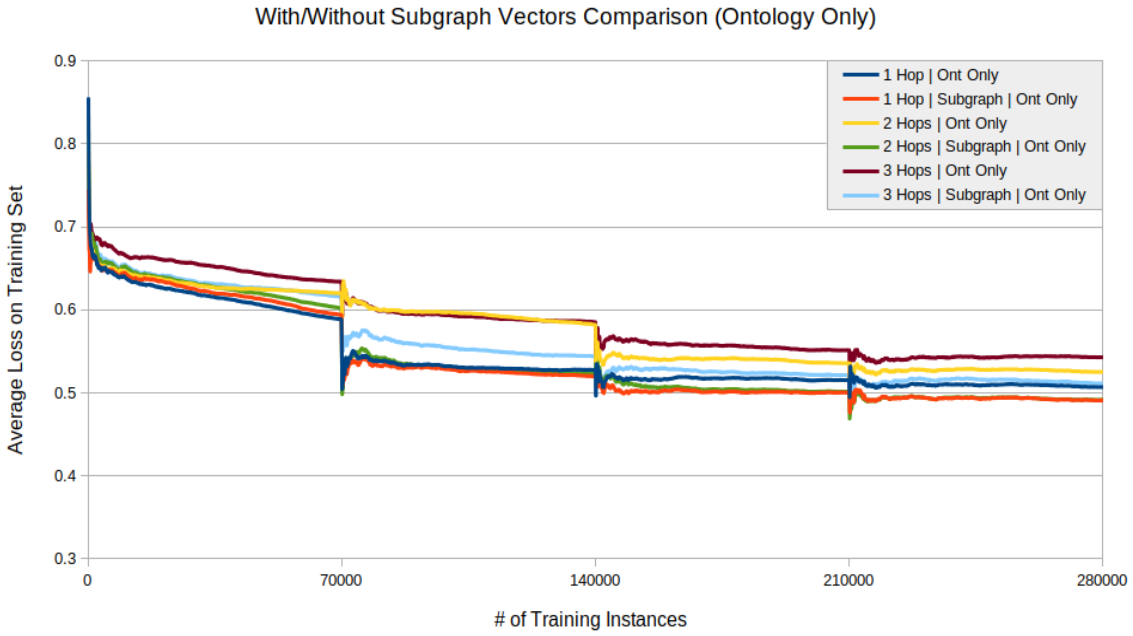

Figure B.3: Training curves (average loss) on split-story cloze-style data for models with/without subgraph context vectors (using only ontology labels), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Subgraph = use subgraph context vectors
Ont Only = only ontology labels are used

	1 Hop	2 Hops	3 Hops
Non-subgraph	.71	.70	.68
Subgraph	.73	.73	.71

Table B.3: Average precision on split-story cloze-style test set for models with/without subgraph context vectors (using only ontology labels), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-subgraph = do not use subgraph context vectors
Subgraph = use subgraph context vectors

Figure B.3 and **Table B.3** compare training loss curves and test precision scores, respectively, for models which either do/do not use additional subgraph context vectors *with concatenative attention and only ontology labels*.

	1+2+3 Avg		First Layer
	Names + Ont	Ont Only	Names + Ont
1 Hop	.004	.012	.008
2 Hops	.018	.023	.012
3 Hops	.014	.025	—

Table B.4: Δ average precision (+) on split-story cloze-style test set when using subgraph context vectors, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = Uses an unweighted average of all three ELMo outputs
First Layer = Uses only the first-layer, context-invariant ELMo output
Names + Ont = Uses both name information and ontology labels
Ont Only = Uses only ontology labels

Table B.4 presents the increase in average precision over the test set when using additional subgraph context vectors for a variety of configurations.

B.2 Contextualized vs. Non-contextualized

As before, we observe that using the contextualized output layers of ELMo offer marginal performance increases over using only the first-layer, context-invariant output.

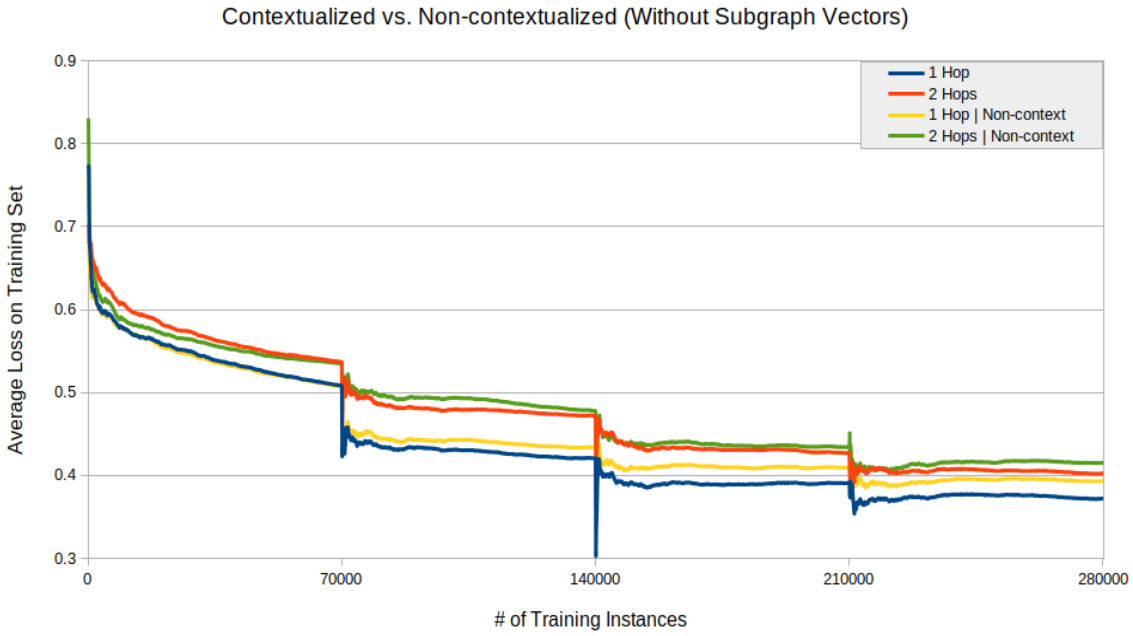

Figure B.4: Training curves (average loss) on split-story cloze-style data for models using either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-context = only the first-layer, context-invariant output of ELMo

	1 Hop	2 Hops
1+2+3 Avg	.77	.76
First Layer	.77	.75

Table B.5: Average precision on split-story cloze-style test set for models using either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = unweighted average of all three ELMo outputs
First Layer = only the first-layer, context-invariant output of ELMo

Figure B.4 and **Table B.5** compare training loss curves and test precision scores, respectively, for models which either use an unweighted average of all three ELMo

outputs ("contextualized") or only the first-layer ELMo output ("non-contextualized") *with concatenative attention.*

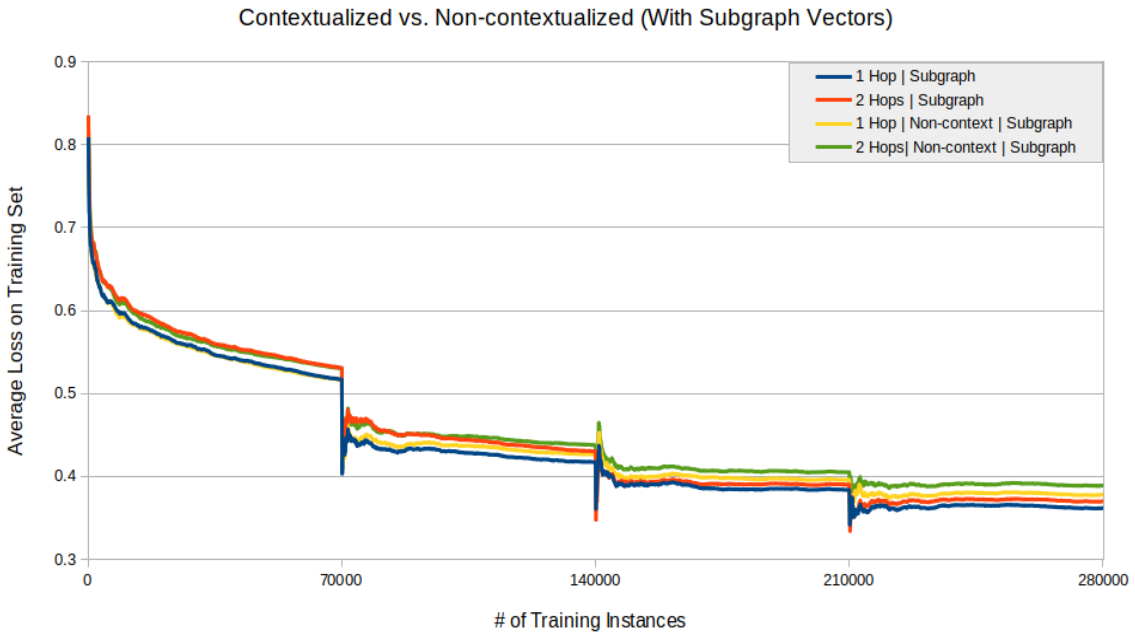

Figure B.5: Training curves (average loss) on split-story cloze-style data for models using subgraph context vectors and either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Non-context = only the first-layer, context-invariant output of ELMo
Subgraph = use subgraph context vectors

	1 Hop	2 Hops
1+2+3 Avg	.79	.78
First Layer	.78	.77

Table B.6: Average precision on split-story cloze-style test set for models using subgraph context vectors and either (i) an unweighted average of all three ELMo outputs ("contextualized") or (ii) only the first-layer ELMo output ("non-contextualized"), where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
1+2+3 Avg = unweighted average of all three ELMo outputs
First Layer = only the first-layer, context-invariant output of ELMo

Figure **B.5** and **Table B.6** compare training loss curves and test precision scores, respectively, for models which either use an unweighted average of all three ELMo outputs ("contextualized") or only the first-layer ELMo output ("non-contextualized") *with concatenative attention and additional subgraph context vectors*.

B.3 Using Name and Ontology Information vs. Ontology Information Only

As before, we observe considerable increases in performance when using both name information and ontology labels during learning, confirming that entity/event name information serves as a critical supplement to ontology labels.

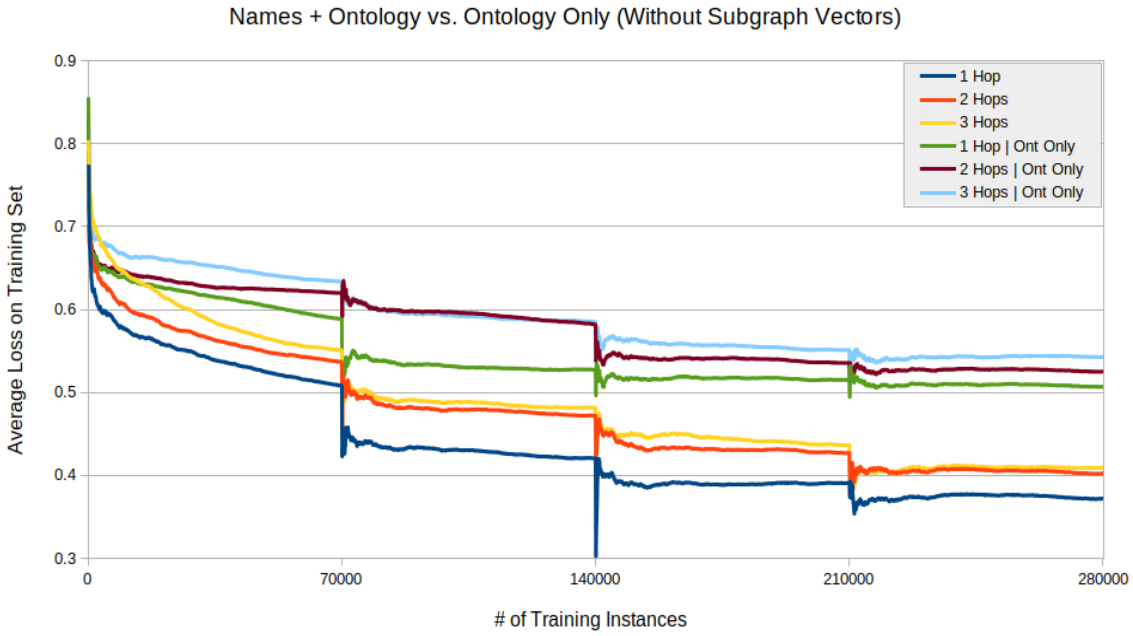

Figure B.6: Training curves (average loss) on split-story cloze-style data for models using either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Ont only = ontology labels alone

	1 Hop	2 Hops	3 Hops
Names + Ont	.77	.76	.76
Ont Only	.71	.70	.68

Table B.7: Average precision on split-story cloze-style test set for models using either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Name + Ont = name information and ontology labels
Ont only = ontology labels alone

Figure B.6 and **Table B.7** compare training loss curves and test precision scores, respectively, for models which either use both name and ontology information or ontology information alone *with concatenative attention*.

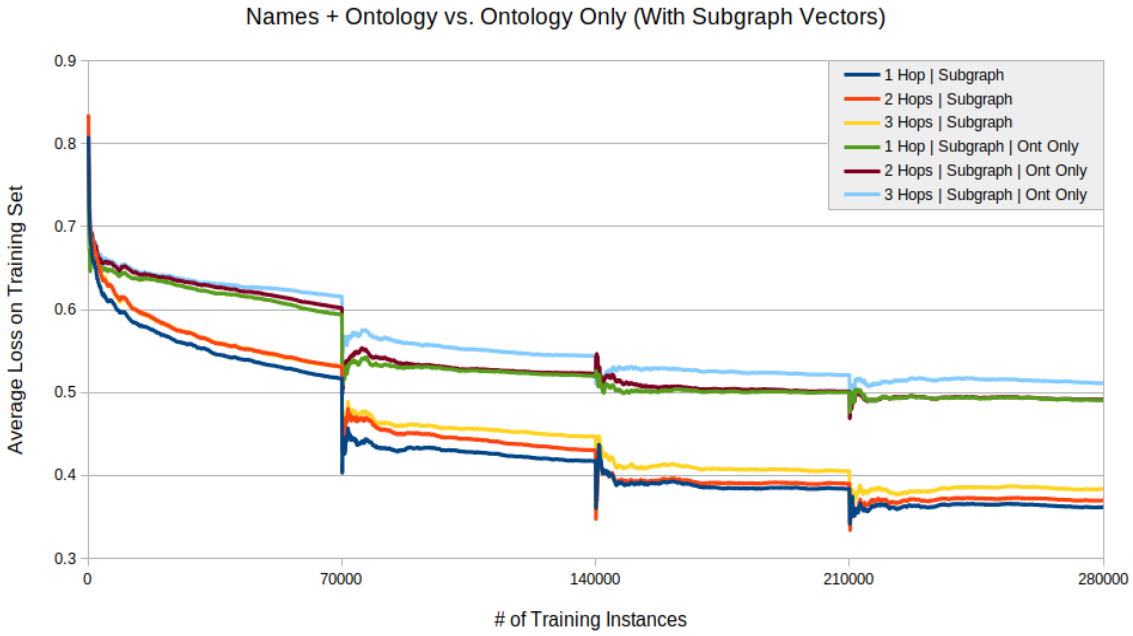

Figure B.7: Training curves (average loss) on split-story cloze-style data for models using subgraph vectors and either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Subgraph = use subgraph context vectors
Ont only = ontology labels alone

	1 Hop	2 Hops	3 Hops
Names + Ont	.79	.78	.78
Ont Only	.73	.73	.71

Table B.8: Average precision on split-story cloze-style test set for models using subgraph context vectors and either (i) name information and ontology labels or (ii) ontology labels alone, where:
x Hop(s) = the set of query statements extends out a maximum of x statement traversals from the bottleneck entity
Name + Ont = name information and ontology labels
Ont only = ontology labels alone

Figure B.7 and **Table B.8** compare training loss curves and test precision scores,

respectively, for models which either use both name and ontology information or ontology information alone *with concatenative attention and additional subgraph context vectors.*

Bibliography

Ron Bekkerman and Koby Crammer. One-class clustering in the text domain. In *Proceedings of EMNLP*, 2008.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2787–2795, Red Hook, NY, USA, 2013. Curran Associates Inc.

Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *Proceedings of ACL-08: HLT*, pages 789–797, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P08-1090`.

Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative schemas and their participants. In *Proceedings of ACL*, page 602–610. Association for Computational Linguistics, 2009.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehnand, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *arXiv preprint arXiv:1312.3005*, 2013.

Benson Chen, Regina Barzilay, and Tommi Jaakkola. Path-augmented graph transformer network, 2019.

Pengxiang Cheng, Alexander Tomkovich, Eric Holgate, Su Wang, and Katrin Erk. The utexas system for tac 2019 sm-kbp task 3: Hypothesis detection with graph convolutional networks. In *Proceedings of Text Analysis Conference*, 2019.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning, 2017.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

Micha Elsner and Eugene Charniak. You talking to me? a corpus and algorithm for conversation disentanglement. In *Proceedings of ACL-08: HLT*, pages 834–842, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P08-1095`.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo, 2020.

Walter R. Fisher. Narration as a human communication paradigm: The case of public moral argument. *Communication Monographs*, 51(1):1–22, 1984. doi:10.1080/03637758409390180.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory, 2009.

Varuna Jayasiri. Proximal policy optimization algorithms–ppo in pytorch (blog post). 2019. URL `https://blog.varunajayasiri.com/ml/ppo_pytorch.html`.

Jyun-Yu Jiang, Francine Chen, Yan-Ying Chen, and Wei Wang. Learning to disentangle interleaved conversational threads with a siamese hierarchical network and similarity ranking. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1812–1822, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi:10.18653/v1/N18-1164. URL `https://www.aclweb.org/anthology/N18-1164`.

Alex Judea and Michael Strube. Incremental global event extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2279–2289, Osaka, Japan, December 2016. The COLING

2016 Organizing Committee. URL https://www.aclweb.org/anthology/C16-1215.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*, 2017.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Proceedings of NeurIPS*, 2000.

Ni Lao, Jun Zhu, Liu Liu, Yandong Liu, and William W Cohen. Efficient relational learning with hidden variable detection. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1234–1242. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/4175-efficient-relational-learning-with-hidden-variable-detection.pdf.

Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 529–539, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D11-1049.

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *In Proceedings of NeurIPS*, 1989.

Manling Li, Ying Lin, Joseph Hoover, Spencer Whitehead, Clare R. Voss, Morteza Dehghani, and Heng Ji. Multilingual entity, relation, event and human value extraction. In *Proceedings of NAACL-HLT*, 2019.

Qi Li, Heng Ji, and Liang Huang. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 73–82, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-1008.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi:10.18653/v1/D18-1362. URL `https://www.aclweb.org/anthology/D18-1362`.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. A joint neural model for information extraction with global features. In *Proceedings of ACL*, 2020.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, 2015.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of EMNLP*, 2017.

Thomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, pages 3111–3119, 2013b.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings NAACL-HLT*, pages 746–751, 2013c.

Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M. Bronstein. Dual-primal graph convolutional networks. arXiv preprint arXiv:1806.00770, 2018.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi:10.3115/v1/D14-1162. URL `https://www.aclweb.org/anthology/D14-1162`.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, 2018a.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of EMNLP*, 2018b.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Nils Reimers and Iryna Gurevych. Alternative weighting schemes for elmo embeddings. In *arXiv preprint arXiv:1904.02954*, 2019.

Roger C. Schank and Robert P. Abelson. Scripts, plans, goals and understanding: An inquiry into human knowledge structures. 1977.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

Sonit Singh. Natural language processing for information extraction, 2018.

Richard S. Sutton and Andrew G. Barto. Policy gradient methods. In *Reinforcement learning: an introduction*, 2018.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of NeurIPS*, 2000.

Wilson L. Taylor. Cloze procedure: A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433, 1953. doi:10.1177/107769905303000401. URL `https://doi.org/10.1177/107769905303000401`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

Su Wang, Eric Holgate, Greg Durrett, and Katrin Erk. Picking apart story salads. In *Proceedings of EMNLP*, 2018.

Su Wang, Greg Durrett, and Katrin Erk. Query-focused scenario construction. In *Proceedings of EMLNP*, 2019.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning (8), pages 229-256*, 1992.

Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

Limin Yao, Sebastian Riedel, and Andrew McCallum. Universal schema for entity type prediction. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, page 79–84, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450324113. doi:10.1145/2509558.2509572. URL `https://doi.org/10.1145/2509558.2509572`.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation, 2018.

Mikuláš Zelinka, Xingdi Yuan, Marc-Alexandre Côté, Romain Laroche, and Adam Trischler. Building dynamic knowledge graphs from text-based games, 2019.